STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

DATA COLLECTIONS AND MONADS

RADU TRÎMBIŢAŞ

Dedicated to Professors Emil Muntean and Ştefan Niţchi

Abstract. This paper tries to introduce data collection using monads. Three data collection types are defined: weak, zero-element and strong. The main result shows that monads lead to data collection types. Then the algebraic properties of data collections are studied and some identities useful to query optimization are given. Another result shows the equivalence of ringad and extended monad with zero. Finally, we treat the type conversion problem.

1. Collections

Data collections are defined in [1]. We shall use other definitions for data collections, and our approach will be categorical. For notions of Category Theory the reader may consult [3, 4, 11]. Also the paper [8] illustrates how Category Theory notions and constructions can be systematically used in Computer Science.

Definition 1.1. *A* **weak data collection** *C is a parametrized abstract data type (with type parameter T)*

$$C(T) = <\ \tau,\ [x],\ ++,\ \mathrm{aggr}\ >$$

where

- *τ is the set of data collections over T;*
- *$[x]$ is the singleton (single element) collection;*
- *$++ : \tau \times \tau \to \tau$ is the concatenation operator of two collections;*
- *aggr is the aggregation operator defined as follows: if $f: T \to S$ and $\oplus : S \times S \to S$ is a binary operation, then $\mathrm{aggr}(f, \oplus) : \tau \to S$ (i.e. $\mathrm{aggr} : (T \to S) \times \tau \to S$) which has the following properties:*

$$\mathrm{aggr}(f, \oplus)([x]) = f(x) \tag{1}$$

Received by the editors: October 15, 1996.

1991 *Mathematics Subject Classification.* 18C15, 68P15.

1991 *CR Categories and Descriptors.* H2.1 [Data base management] Logical Design – data models, H2.4 [Data base management] Systems – query processing.

$$\text{aggr}(f,\oplus)(x{+\!\!+}y) = \text{aggr}(f,\oplus)(x) \oplus aggr(f,\oplus)(y), \quad x,y \in \tau. \tag{2}$$

Definition 1.2. *A* **zero-element data collection** C *is a parametrized abstract data type (with type parameter T)*

$$C(T) \ = < \ \tau, \ [], \ [x], \ {+\!\!+}, \ \text{aggr} \ >$$

where

- $< \ \tau, \ [x], \ {+\!\!+}, \ \text{aggr} \ >$ *is a weak data collection;*
- $[]$ *is the empty collection and*

$$\forall x \in \tau \qquad x{+\!\!+}[] = []{+\!\!+}x = x. \tag{3}$$

- aggr *is the aggregation operator defined as follows: if $f : T \to S$ and $\oplus : S \times S \to S$ is a binary operation with neutral element u, then* $\text{aggr}(f,\oplus) : \tau \to S$ *(i.e.* $\text{aggr} : (T \to S) \times \tau \to S$ *) which verifies (1), (2) and*

$$\text{aggr}(f,\oplus)([]) \ = \ u. \tag{4}$$

Definition 1.3. *A* **strong data collection** *is a zero-element data collection such that the concatenation is associative, that is*

$$\forall x,y,z \in \tau \qquad (x{+\!\!+}y){+\!\!+}z = x{+\!\!+}(y{+\!\!+}z). \tag{5}$$

The notion of strong collection is similar to monoid collection defined in [5, 6, 7], but there the approach is not categorical.

Remark 1.4. Each strong data collection is also a zero-element data collection; each zero-element data collection is also a weak data collection.

In the sequel $[x_1, \ \ldots, \ x_n]$ will denote the finite collection having the elements $x_1, \ldots x_n$.

2. Monads and data collection

Definition of data types through monads is a usual method in Functional Programming. As each adjunction induces a monad (Huber's theorem see [3, 4, 10]) and each monad leads to adjunctions in several ways, Eilenberg-Moore's and Kleisli's construction being the most known ([3, 4, 10]), the results must resemble those obtained through adjunctions.

Definition 2.1. *A* **monad** *(or* **triple***)* $\mathbf{T}=(T,\eta,\mu)$ *on category $\mathcal{C}$ is an endofunctor $T : \mathcal{C} \to \mathcal{C}$ together with two natural transformations $\eta : id_{\mathcal{C}} \to T$ and*

$\mu : T^2 \to T$ *such that the following diagrams commute:*

$$\begin{tikzcd}
T^3 \arrow[r, "\mu T"] \arrow[d, "T\mu"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
T \arrow[r, "\eta T"] \arrow[dr, "id"'] & T^2 \arrow[d, "\mu"] & T \arrow[l, "T\eta"'] \arrow[dl, "id"] \\
& T &
\end{tikzcd}$$

(*mon1*) (*mon2*)

In these diagrams T^n *means* T *composed by himself* n *times.* ■

For examples and properties of monads see [3, 10] and for applications of monads in Computer Science see [12, 13, 14, 15].

Another equivalent definition [9, 14] is:

Remark 2.2. A **monad** on $\mathcal{C}$ is the system $(T, \eta, -^*)$ where T and η are as above, and * is the iteration operator (the Kleisli star) defined as follows: if $f : A \to TB$ then $f^* : TA \to TB$ and it satisfies

$$\begin{aligned}
(\eta_S)^* &= id_{TA} \\
f^* \circ \eta_A &= f \\
g^* \circ f^* &= (g^* \circ f)^*. \ ■
\end{aligned}$$

Let's see how data collections, aggregation operators and their properties may be obtained through monads, using the Eilenberg-Moore's construction.

Definition 2.3. *If* $\mathbf{T}=(T, \eta, \mu)$ *is a monad on* $\mathcal{C}$, *a* $\mathbf{T}$*-algebra is a pair* (A, a), *where* $A \in Ob\mathcal{C}$ *and* $a : TA \to A$ *such that the following diagrams commute:*

(6)
$$\begin{tikzcd}
A \arrow[r, "\eta A"] \arrow[dr, "id_A"'] & TA \arrow[d, "a"] \\
& A
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2A \arrow[r, "Ta"] \arrow[d, "\eta A"'] & TA \arrow[d, "a"] \\
TA \arrow[r, "a"'] & A
\end{tikzcd}$$

The arrow a is the structure-mapping of the algebra. ■

Definition 2.4. *A mapping $f : (A, a) \to (B, b)$ is a* $\mathbf{T}$*-algebra homomorphism if the diagram*

$$\begin{tikzcd} TA \arrow[r, "Tf"] \arrow[d, "a"'] & TB \arrow[d, "b"] \\ A \arrow[r, "f"'] & B \end{tikzcd} \tag{7}$$

commutes. ■

$\mathbf{T}$-algebra and $\mathbf{T}$-algebra homomorphisms form a category $\mathcal{C}^T$. We define $U^T : \mathcal{C}^T \to \mathcal{C}$ by $U^T(A, a) = A$ and $U^T(f) = f$ and $F^T : \mathcal{C} \to \mathcal{C}^T$ by $F^T(a) = (TA, \mu_A)$ and $F^T(f) = T(f)$. The functor F^T is the left adjoint of U^T, and the unit and counit of the adjunction are $\eta^T = \eta$ and $\varepsilon^T(A, a) = a : F^T U^T(A, a) \to (A, a)$ respectively (the Eilenberg-Moore's construction). We shall consider only monads over Cartesian closed categories and $\mathcal{C}$ will be $\mathcal{T}$ (the category of types).

Definition 2.5. *The quadruple* $\mathbf{T}$=$(T, \eta, \mu, conc)$ *is an* **extended monad** *on* $\mathcal{C}$ *if:*

1. (T, η, μ) *is a monad on* $\mathcal{C}$*;*
2. $conc : T \times T \to T$ *is a natural transformation;*
3. $\forall B \in Ob\mathcal{C}$ $\mu_B \circ conc_B = conc_{TB} \circ (\mu_B \times \mu_B)$, *that is the following diagram*

$$\begin{tikzcd} T^2B \times T^2B \arrow[r, "conc_{TB}"] \arrow[d, "\mu_B \times \mu_B"'] & T^2B \arrow[d, "\mu_B"] \\ TB \times TB \arrow[r, "conc_B"'] & TB \end{tikzcd} \tag{8}$$

commutes. ■

Proposition 2.6. For each monad $\mathbf{T}$=(T, η, μ) and for each natural transformation $conc : T \times T \to T$ it holds

$$conc_B = \mu_B \circ conc_{TB} \circ (T\eta_B \times T\eta_B).$$

Proof. Using naturality of conc and monad definition we have

$$\mu_B \circ conc_{TB} \circ (T\eta_B \times T\eta_B) = \mu_B \circ T\eta_B \circ conc_B = conc_B. ■$$

Theorem 2.7. *The extended monad* $\mathbf{T}$=$(T, \eta, \mu, conc)$ *determines a weak data collection.*

Proof. For a type X we build

$$C(X) \;=<\; \tau,\ [x],\ ++,\ \mathbf{aggr} \;>$$

such that τ is $T(X)$, $[x]$ is $\eta_X(x)$ and ++ is *conc*. The aggregation operator is defined as follows: let $f : A \to B$ where (B,b) is a **T**-algebra. If $\alpha, \beta \in B$ we define $\alpha \oplus \beta = b[\alpha, \beta]$, where

$$[\alpha, \beta] = conc_B([\alpha], [\beta]) = conc_B(\eta_B(\alpha), \eta_B(\beta)).$$

Now we define $\mathbf{aggr}(f, \oplus) = b \circ Tf$. Let's prove the above operator satisfies (1) and (2). As $\eta : id \to T$ is a natural transformation and (B, b) is a **T**-algebra we have the following commutative diagrams

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\eta_A"'] & B \arrow[d, "\eta_B"] \arrow[dr, "id_B"] & \\
TA \arrow[r, "Tf"'] & TB \arrow[r, "b"'] & B
\end{tikzcd}$$

that is

$$Tf \circ \eta_A = \eta_B \circ f \tag{9}$$

$$b \circ \eta_B = id_B \tag{10}$$

From (9) and (10) we have

$$b \circ Tf \circ \eta_A = b \circ \eta_B \circ f = id_B \circ f = f \tag{11}$$

that is (1).

We have successively the following equalities:

$$\begin{aligned}
& b \circ Tf \circ conc_A = && \text{(naturality of conc)}\\
&= b \circ conc_B \circ (Tf \times Tf) && \text{(monad)}\\
&= b \circ conc_B \circ (\mu_B \times \mu_B) \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf) && \text{(extended monad)}\\
&= b \circ \mu_B \circ conc_{TB} \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf) && \text{((}B, b\text{) T-algebra)}\\
&= b \circ Tb \circ conc_{TB} \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf) && \text{(naturality of conc)}\\
&= b \circ conc_B \circ (Tb \times Tb) \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf) && \\
&= b \circ conc_B \circ [(Tb \circ \eta_{TB} \circ Tf) \times (Tb \circ \eta_{TB} \circ Tf)] && \text{(naturality of } \eta\text{)}\\
&= b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (\eta_{TB} \circ (b \circ Tf))] &&
\end{aligned}$$

Thus we have

$$b \circ Tf \circ conc_A = b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (b \circ Tf))].$$

The above relation applied to (x, y) where $x, y \in \tau$ gives us

$$(b \circ Tf \circ conc_A)(x, y) = (b \circ Tf)(x{+}{+}y) = \mathbf{aggr}(f, \oplus)(x{+}{+}y)$$

and

$$\begin{aligned} & b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (\eta_{TB} \circ (b \circ Tf))(x, y) \\ = {} & b \circ conc_B[(\eta_{TB} \circ (b \circ Tf))(x), (\eta_{TB} \circ (b \circ Tf))(y)] \\ = {} & b \circ conc_B(\eta_{TB}(\mathrm{aggr}(f, \oplus)(x)), \eta_{TB}(\mathrm{aggr}(f, \oplus)(y))) \\ = {} & b \circ conc_B([\mathrm{aggr}(f, \oplus)(x)], [\mathrm{aggr}(f, \oplus)(y)]) \\ = {} & \mathrm{aggr}(f, \oplus)(x) \oplus \mathrm{aggr}(f, \oplus)(y) \end{aligned}$$

that is (2). ■

Proposition 2.8. It holds

$$\mathrm{aggr}(\eta, ++) = id_T. \tag{12}$$

Proof. Because (TA, μ_A) is a **T**-algebra and from monads definition we have for $x \in TA$

$$\mathrm{aggr}(\eta, ++) = (\mu_a \circ T\eta_A)(x) = id_{TA}(x). ■$$

Proposition 2.9. If $f : A \to B$ and $h : B \to C$ is a **T**-algebra homomorphism then

$$h \circ b \circ Tf = c \circ T(h \circ f).$$

Proof. We have

$$\begin{array}{lcl} h \circ b \circ Tf & = & (h\ \mathbf{T}\text{-algebra homomorphism}) \\ c \circ Th \circ Tf & = & (T\ \text{functor} \\ c \circ T(h \circ f). ■ & & \end{array}$$

Corollary 2.10.

$$h \circ aggr(f, \oplus) = aggr(h \circ f, \otimes). ■$$

Definition 2.11. *A* **quad** *or* **monad with zero** *is a monad with a family of functions $zero_{A,B}$ which satisfies*

$$\begin{aligned} Tf \circ zero_{C,A} &= zero_{A,B} \circ g \\ \mu_B \circ zero_{A,TB} &= zero_{A,B} \\ \mu_B \circ T(zero_{A,B} &= zero_{TA,B} \end{aligned}$$

for each $g : C \to A$ and $f : A \to TB$, that is the following diagrams commute:

$$\begin{tikzcd} C \arrow[r, "zero_{C,A}"] \arrow[d, "g"'] & TA \arrow[d, "Tf"] \\ A \arrow[r, "zero_{A,B}"'] & TB \end{tikzcd}$$

(*quad1*)

$$\begin{tikzcd} A \arrow[r, "zero_{A,TB}"] \arrow[dr, "zero_{A,B}"'] & T^2B \arrow[d, "\mu_B"] & TA \arrow[l, "T(zero_{A,B})"'] \arrow[dl, "zero_{TA,B}"] \\ & TB & \end{tikzcd}$$

(*quad2*) ■

Remark 2.12. We have an equivalent definition stating for the family $zero_{A,B} : A \to TB$ the conditions (see [9]):

$$\begin{aligned} zero_{A,B} \circ g &= zero_{C,B} \\ (zero_{A,B})^* &= zero_{TA,B} \\ f^* \circ zero_{C,A} &= zero_{C,B}. \blacksquare \end{aligned}$$

Definition 2.13. *The quadruple $(T, \eta, -^*, ++)$ is a* **semiringad** *if $(T, \eta, -^*)$ is a monad, and the concatenation $++$ verifies*

$$f^*(x{+\!+}y) = f^*(x){+\!+}f^*(y)$$

for each $f : A \to TA$. $\blacksquare$

Definition 2.14. *A* **ringad** *is a semiringad $(T, \eta, -^*, ++)$ which is also a quad, and the zero of quad is the neutral element for concatenation.*

The notion of ringad was introduced by Philip Wadler and colab. They consider it as a natural framework for data collection definition and study.

Semiringad and extended monad are equivalent notions.

Theorem 2.15. *The quadruple $\mathbf{T} = (T, \eta, -^*, ++)$ is an extended monad iff it is a semiringad.*

Proof.
(Necessity) If $f : A \to TB$ we have

$$F^* = \mu_B \circ Tf \tag{13}$$

$$\begin{aligned} f^* \circ conc_A = \mu_B \circ Tf \circ conc_A &= && \text{(from (13))} \\ \mu_B \circ conc_{TB} \circ (Tf \times Tf) &= && \text{(extended monad)} \\ conc_B \circ (\mu_B \times \mu_B) \circ (Tf \times Tf) &= && \\ conc_B \circ (\mu_B \times Tf) \circ (\mu_B \times Tf) &= && \text{(from (13))} \\ conc_B \circ (f^* \times f^*). & && \end{aligned}$$

(Sufficiency)
From $f^* \circ conc_A = conc_B \circ (f^* \times f^*)$ (semiringad property) we obtain

$$\begin{aligned} & = && \text{(naturality of conc)} \\ \mu_B \circ Tf \circ conc_A & = && \text{(naturality of conc)} \\ \mu_B \circ conc_{TB} \circ (Tf \times Tf) & && \\ conc_B \circ (\mu_B \times \mu_B) \circ (Tf \times Tf). & && \end{aligned}$$

Because the last equality holds for each $f : A \to TB$, choosing f such that $(Tf \times Tf)$ be an epimorphism we have

$$\mu_B \circ conc_{TB} = conc_B \circ (\mu_B \times \mu_B). \blacksquare$$

Remark 2.16. Each ringad is an extended monad which is also a quad and the zero of quad is neutral element for concatenation. $\blacksquare$

Theorem 2.17. *Each ringad determines a zero-element data collection.*

Proof. From remark 2.16, (1) and (2) are true. The proof proceeds as in theorem 2.7 putting $b([\,]) = u$. From ringad definition we have

$$\operatorname{aggr}(f, \oplus)(x) = \operatorname{aggr}(f, \oplus)(x{+}{+}[\,]) = \operatorname{aggr}(f, \oplus)(x) \oplus \operatorname{aggr}(f, \oplus)([\,])$$

which implies $\operatorname{aggr}(f, \oplus)([\,]) = u$, that is (4). ■

Corollary 2.18. Each ringad whose concatenation is associative determines a strong data collection. ■

3. Examples

Example 3.1. If T is the functor which maps a type (set) A to the free algebra with one binary operation (subject to no restriction) generated by A we obtain labeled rooted ordered binary trees. The mapping η maps $x \in A$ to the single-node labeled with x tree, the multiplication μ maps a tree whose labels are trees in TA to the tree obtained attaching to each node the tree having as name the label of that node. The concatenation of trees t_1 and t_2 is the tree having t_1 as left subtree and t_2 as right subtree. The empty tree is the zero of the monad. Alternatively, this collection type may be obtained using an adjunction from the category $\mathcal{B}$ of algebras with one binary operation to the category $\mathcal{T}$ of types. ■

Example 3.2. Let $T : \mathcal{T} \to \mathcal{T}$ be the functor which maps the type(set) A to Kleene's closure A^* of A (the underlying set of free-monoid generated by A) and the function $f : A \to B$ to the unique extension $f^* : A^* \to B^*$. Let $\eta A : A \to A^*$ be the application which maps a to the word $[a]$ (having length equal to 1) and $\mu A : A^{**} \to A$ which take a word of words $[s_1, \ldots, s_k]$ and maps it to the concatenation $s_1 \ldots s_k$ in A^* (obtained removing inner parentheses). The concatenation $concA : A^* \times A^* \to A^*$ is as usual and the null word is his neutral (unit) element. The mappings $\eta : id \to T$, $\mu : T^2 \to T$ and $conc : T \times T \to T$ are natural transformations and $(T, \eta, \mu, conc, zero)$ is a ringad ($zero_{A,B}$ is the null word). Thus we have the list collection type.

This ringad can also be obtained from the monad determined by forgetful functor U : **Mon** $\to \mathcal{T}$ and his left adjoint (the free functor). ■

Example 3.3. Let $T : \mathcal{T} \to \mathcal{T}$ be the functor which maps the type(set) X to the underlying set of commutative free-monoid generated by X. TX is the set of equivalence classes of words made from symbols $x \in X$; a word which contain a segment xy is equivalent to the word obtained replacing all occurrences of xy by yx. Let $[w]$ be the equivalence class of w. The application ηX maps x to $[x]$ and μX maps a word of words in TX to a word in X removing inner parentheses. Concatenation of two words is the equivalence class of the word obtained through usual concatenation. Thus we obtain a ringad which determines the multiset(bag) collection type.

Equivalently, this monad is induced by the adjunction $< U, F, \eta, \mu >$ where U : **CMon** $\to \mathcal{T}$ is the forgetful functor and $F : \mathcal{T} \to$ **CMon** is his left adjoint (extended with concatenation and null word). ■

Example 3.4. Let be $P : \mathcal{T} \to \mathcal{T}$(or more generally P : **Set** $\to \mathcal{S}et$) given by $P(X) = \mathcal{P}(x)$(the powerset of X) and for $f : X \to Y$, $\mathcal{P}f : \mathcal{P}(X) \to \mathcal{P}(Y)$, $(\mathcal{P}f)(X) = f(X)$.
We define $\eta X : X \to \mathcal{P}(X)$, $\eta X(x) = x$ and $\mu X : \mathcal{P}\mathcal{P}(X) \to \mathcal{P}(X)$ which maps a set of sets to their union. The concatenation is the usual union and the zero is $\emptyset$. We have a ringad which generates the set collection type. The equivalent adjunction is determined by the underlying functor U : **UCSL** $\to \mathcal{T}$ from the category of upper complete semilattices to $\mathcal{T}$ (or **Set**) and his left adjoint (with corresponding extension). ■

Remark 3.5. In fact, every equationally defined category of one sorted algebraic structures is equivalent to the category of Eilenberg-Moore algebras for some monad in **Set** (Linton's theorem [3]). ■

Example 3.6. (Arrays) The example is inspired from [2] which treats it in the context of computing the Fast Fourier Transform of a vector stored in a database. An **array** is like a list having in front its size. The unit of monad is a one element array, μ is the flatten(linearization) of a two dimensional array, the zero is the empty array, and the concatenation is the juxtaposition. ■

4. Type conversion

Type conversion would be an answer to the problem of managing simultaneous several collection type.

Definition 4.1. *Let* **T**$=(T, \eta, \mu)$ *and* **T'**$=(T', \eta', \mu')$ *two monads on the same category* $\mathcal{C}$. *A* **monad homomorphism** α : **T** $\to$ **T**$'$ *is a natural transformation* $\alpha : T \to T'$ *such that the following diagrams commute:*

$$\begin{tikzcd}
id \arrow[d, "\eta"'] \arrow[dr, "\eta'"] & \\
T \arrow[r, "\alpha"'] & T'
\end{tikzcd}$$

$(monadhom1)$

$$\begin{tikzcd}
T^2 \arrow[r, "\alpha^2"] \arrow[d, "\mu"'] & T'^2 \arrow[d, "\mu'"] \\
T \arrow[r, "\alpha"'] & T'
\end{tikzcd}$$

$(monadhom2)$

where $\alpha^2 = T'\alpha \circ \alpha T' = \alpha T' \circ T\alpha$. ■

Definition 4.2. *Let* **T**$=(T,\eta,\mu,conc)$ *and* **T'**$=(T',\eta',\mu',conc')$ *extended monads on the same category* C. *An* **extended monad homomorphism** *from* **T** *to* **T'** *is a monad homomorphism* α : **T** $\rightarrow$ **T'** *such that the following diagram commutes:*

$$\begin{tikzcd}
T \times T \arrow[r, "conc"] \arrow[d, "\alpha \times \alpha"'] & T \arrow[d, "\alpha"] \\
T' \times T' \arrow[r, "conc'"'] & T'
\end{tikzcd}$$

(mexthom) ■

Definition 4.3. *Let* **T**$=(T,\eta,\mu,zero)$ *and* **T'**$=(T',\eta',\mu',zero')$ *two quads on the same category* C. *A* **quad homomorphism** *or a* **monad with zero homomorphism** *from* **T** *to* **T'** *is a monad homomorphism* α : **T** $\rightarrow$ **T'** *such that the following diagram commutes:*

$$\begin{tikzcd}
A \arrow[d, "zero_{A,B}"'] \arrow[dr, "zero'_{A,B}"] & \\
TB \arrow[r, "\alpha_B"'] & T'B
\end{tikzcd}$$

(quadhom) ■

Definition 4.4. *Let* **T**$=(T,\eta,\mu,conc,zero)$ *and* **T'**$=(T',\eta',\mu',conc',zero')$ *two ringads on the same category* C. *A* **ringad homomorphism** *from* **T** *to* **T'** *is an extended monad homomorphism* α : **T** $\rightarrow$ **T'** *which is also a quad homomorphism.* ■

Let $C(T) =< \tau, [x], ++, \text{aggr} >$ and $C'(T) =< \tau', [x]', ++', \text{aggr}' >$ be two data collection types over the same type T, defined by the extended monads **T**$=(T,\eta,\mu,conc)$ and **T'**$=(T',\eta',\mu',conc')$.

Let's solve the problem of conversion from $C(T)$ to $C'(T)$.

Let $[x_1,\dots,x_n] \in C(T)$. We have from proposition 2.8

$$[x_1,\dots,x_n] = [x_1] ++ \dots ++ [x_n].$$

Usually, such a conversion is done as follows

(14)
$$\alpha([x]) = [x]'$$

(15)
$$\alpha([x_1,\dots,x_n]) = [x_1]' ++' \dots ++' [x_n]' = [x_1,\dots,x_n]'$$

Proposition 4.5. The mapping α defined as above is an extended monad homomorphism.

Proof.

(a) Let's prove the naturality of α. Let $[x_1, \ldots, x_n] \in TA$. We have $Tf([x_1, \ldots, x_n]) = [f(x_1), \ldots, f(x_n)]$. Also,

$$\begin{aligned}(\alpha B \circ Tf)([x_1, \ldots, x_n]) &= \alpha B([f(x_1), \ldots, f(x_n)] = \\ &= [f(x_1), \ldots, f(x_n)]'\end{aligned}$$

and

$$\begin{aligned}(T'f \circ \alpha A)([x_1, \ldots, x_n]) &= (T'f)[x_1, \ldots, x_n]' = \\ &= [f(x_1), \ldots, f(x_n)]',\end{aligned}$$

that $\alpha B \circ Tf = T'f \circ \alpha A$.

(b) The commutativity of $(monadhom1)$ follows immediately from the definition of α:

$$(\alpha \circ \eta)(x) = \alpha([x]) = [x]' = \eta'(x).$$

(c) Let's prove the commutativity of (monadhom2).

Let $[[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]] \in T^2A$.

$$\begin{aligned}(\alpha \circ \mu)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ \alpha\left([x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,1}, \ldots, x_{l,k_l}]\right) &= \\ [x_{1,1}]' +\!\!+' \cdots +\!\!+' [x_{1,k_1}]' +\!\!+' \cdots +\!\!+' [x_{l,1}] +\!\!+' \cdots +\!\!+' [x_{l,k_l}]' &= \\ [x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,k_l}]' & \end{aligned} \tag{16}$$

On the other hand:

$$\begin{aligned}\alpha^2\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ (\alpha T' \circ T\alpha)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ \alpha T'\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]'\right) &= \\ \left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]'\right) & \end{aligned}$$

and

$$\begin{aligned}(\mu' \circ alpha^2)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots x_{l,k_l}]]\right) &= \\ \mu'\left([[x_{1,1}, \ldots, x_{1,k_1}]', \ldots [x_{l,1}, \ldots x_{l,k_l}]']'\right) &= \\ [x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,k_l}]' & \end{aligned} \tag{17}$$

Relations (16) and (17) imply the desired commutativity.

(d) Let's prove now the commutativity of (mexthom).

Let $(x, y) = ([x_1, \ldots, x_k], [y_1, \ldots, y_l]) \in TA \times TA$. We have

$$(\alpha \circ conc)(x, y) = \alpha([x_1, \ldots, x_k, y_1, \ldots y_l]) = [x_1, \ldots, x_k, y_1, \ldots y_l]'$$

and

$(conc' \circ (\alpha \times \alpha))(x, y) = conc'([x_1, \ldots, x_k]', [y_1, \ldots, y_l]') = [x_1, \ldots, x_k, y_1, \ldots y_l]'$

that is (mexthom). ■

Let's extend now the conversion to zero-element collection:

$$\begin{aligned} \alpha([\,]) &= [\,]' \\ \alpha([x]) &= [x]' \\ \alpha([x_1, \ldots, x_n]) &= [x_1]' +\!\!+' \cdots +\!\!+' [x_n]' = [x_1, \ldots, x_n]' \end{aligned} \tag{18}$$

Proposition 4.6. α defined above is a ringad homomorphism.

Proof. The previous proposition implies α is an extended monad homomorphism and (18) implies the commutativity of (quadhom). ■

References

[1] Catriel Beeri, Yoram Kornatzky – Algebraic Optimization of Object-Oriented Query Languages, *Report* 91-6, Hebrew University of Jerusalem, Israel, 1991.

[2] P. Buneman – The Fast Fourier Transform as a Database Query, *Technical Report* MS-CIS-93-27/L&C 60, University of Pennsylvania, March, 1993.

[3] Michael Barr, Charles Wells – *Toposes, Triples and Theories*, Springer Verlag, Berlin, Heildelberg, Tokyo, 1985.

[4] Michael Barr, Charles Wells – *Category Theory for Computing Science*, Prentice-Hall, 1990.

[5] L. Fegaras – A Uniform Calculus for Collection Types, *Technical Report* No. CS/E 94-030, OGI, December, 1994.

[6] L. Fegaras, D. Maier – Towards an Effective Calculus for Object-Oriented Query Languages, *ACM SIGMOD International Conference on Management of Data*, San-Jose, May, 1995.

[7] L. Fegaras, D. Maier – An Algebraic Framework for Physical OODB Design, *5th International Workshop on Database Programming Languages*, Gubbio, Italy, 1995.

[8] Joseph A. Goguen – A categorical manifesto, *Math. Struct. in Comp. Science*, vol. 1, pp. 49-67, 1991.

[9] E. G .Manes – *Algebraic Theories*, Springer-Verlag, Berlin, 1976.

[10] Saunders MacLane – *Categories for the Working Mathematicians*, Springer-Verlag, 1971.

[11] Benjamin C. Pierce – A Taste of Category Theory for Computer Scientist, *Research Report* CMU-CS-88-203, Carnegie-Mellon University, Pittsburgh, 1988.

[12] P. Wadler – The essence of functional programming, *19th Annual Symposium on Principles of Programming Languages*, Santa Fe, New Mexico, January, 1992.

[13] P. Wadler – Comprehending monads, *Math. Struct. in Comp. Science*, vol. 2 (1992), pp. 461-493.

[14] P. Wadler – Monads and composable continuation, *LISP and Symbolic Computation*, 7, pp. 39-56, 1994.

[15] P. Wadler – How to Declare an Imperative, *ILPS 95*, J. Lloyd (ed), 1995.

"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, RO-3400 Cluj-Napoca, Romania

E-mail address: tradu@math.ubbcluj.ro