STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 1, March 1996

ADJUNCTIONS AND DATA COLLECTIONS

R. TRÎMBIŢAŞ

Dedicated to Professor Ştefan I. Niţchi

Abstract. We introduce data collections through adjoints. Weak, zero-element and strong data collection types and aggregation operators are introduced. Ones prove that adjoints lead to data collections. Their properies are studied and we prove some algebraic properties useful to database query optimization. Thus we find again classical colection types: lists, trees, sets.

1. Introduction

Data collections play a central role in Database Theory. A database programming language (DBPL) can model application domains most naturally if it has several collection (bulk) types, e.g., lists, sets trees and so on. The first demonstration that a collection type could be accommodated satisfactorily in a strongly typed programming language was Pascal-R [15]. The collection type concept supports two essential activities. First, it allows regularity in structure to be described. Second, it supports powerful and succinct notations for computing with such regular structures. The requirements for collection data types are summarized and argued about in [2]. A type system for collections tends to be rich and complex, since constructs must be provided to declare, construct, inspect and update instances of each collection type. One needs to control the complexity of such type systems by exploiting operations and properties common to a variety of collection types. We think Category Theory is a possible background to describe regularities, developing algebras for collection types and improving efficiency.

This paper tries to introduce definitions for data collections and study and model them using adjoints. We shall show how data collections can be modelled through adjunctions and how adjunctions lead to collection operators whose algebraic properties are studied. These properties may be used for query optimization.

Received by the editors: October 20, 1996.
1991 *Mathematics Subject Classification.* 18A40, 68P15.
1991 *CR Categories and Descriptors.* H.2.1 [Database Management]: Logical Design - data models; H.2.3 [Database Management]: Languages - data manipulation languages, query languages.

2. Basic Notions

In [4] data collections are defined as follows:

Definition 2.1. *A* **data collection** *is a parametrized abstract data type (with type parameter T) of type $C(T)$ with the following operations:*

- *A family of data constructors $C^n : T^n \to C(T)$ with the arity n, for each $n \geq 0$.*
- *Functions over data collections of type $C(T)$.*
- *Observers –*
  - *(a) selectors: functions from data collections of type $C(T)$ to type T;*
  - *(b) predicates over $C(T)$.*

The algebra of the data collections, that is the semantics of the data constructors and other operations, is defined by a set of first-order Horn clause axioms over the data constructors, functions and observers.

We shall use other definitions for data collections, and our approach will be categorical. For notions of Category Theory the reader may consult [7, 8, 14]. Also the paper [12] illustrates how Category Theory notions and constructions can be systematically used in Computer Science.

Definition 2.2. *A* **weak data collection** *C is a parametrized abstract data type (with type parameter T)*

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

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 $\text{aggr}(f, \oplus) : \tau \to S$ (i.e. $\text{aggr} : (T \to S) \times \tau \to S$) has the following properties:*

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

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

Definition 2.3. *A* **zero-element data collection** *C is a parametrized abstract data type (with type parameter T)*

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

where

- $< \ \tau, \ [x], \ +\!\!+, \ \mathrm{aggr} \ >$ *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* $\mathrm{aggr}(f, \oplus) : \tau \to S$ *(i.e.* $\mathrm{aggr} : (T \to S) \times \tau \to S$ *) which verifies (1), (2) and*

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

Definition 2.4. *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 definition of strong data collection requires $< \tau, \ +\!\!+, \ [] >$ be a monoid. A good model for strong collections is the free monoid over T (directly applicable to lists). The notion of strong collection is similar to monoid collection defined in [9, 10, 11], but there the approach is not categorical.

Remark 2.5. 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 $x_1, \ldots x_n$. Some data collections properties from [4] are expressed using concatenation:

- **permutability** – a data collection type is permutable if
  x ++y = y ++x
- **duplicate elimination** – a data collection type eliminates duplication if ++is idempotent, that is
  x ++x = x

- **null value elimination** – a zero-element data collection type eliminates null values if

  x ++ []= x.

Remark 2.6. Each zero-element data collection elimines null-values.

3. Data collections through adjunctions

Let $\mathcal{T}$ be the category of types (whose morphisms are usual functions), $\mathcal{A}$ an arbitrary category and $< F, U, \eta, \varepsilon >$ an adjunction where $U : \mathcal{A} \to \mathcal{T}$, $F : \mathcal{T} \to \mathcal{A}$ are functors such that $F \dashv U$, and η and ε are the unit and respectively the counit of the adjunction. Let us suppose $\mathcal{A}$ is a category of sets with structure having at least one binary operation. Let $U : \mathcal{A} \to \mathcal{T}$ be the forgetful functor. It has a left adjoint $F : \mathcal{T} \to \mathcal{A}$. If $T \in \mathrm{Ob}\mathcal{T}$ then F(T) is the free-algebra generated by T.

Theorem 3.1. *If $U : \mathcal{A} \to \mathcal{T}$ is the forgetful functor and F is his left adjoint, then the adjunction $< F, U, \eta, \varepsilon >$ determines a weak data collection.*

Proof. If $F \dashv U$ then there exist $\eta : id_{\mathcal{T}} \to UF$ (the unit of the adjunction) such that $\forall X \in \mathrm{Ob}\mathcal{T}$, $\forall Y \in \mathrm{Ob}\mathcal{T}$, $\forall f : X \to UY$ $\exists! f^{\#} \in Hom_{\mathcal{A}}(F(X), Y)$ such that the following diagram commutes

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & UF(X) \arrow[d, dashed, "U(f^{\#})"] \\ & U(Y) \end{tikzcd} \tag{6}$$

(see [8, 14, 13]).

For each $T \in Ob\mathcal{T}$, $F(T)$ is the free-algebra with one binary operation generated by T. We shall take $\tau = UF(T)$, and the concatenation will be the binary operation over $F(T)$. The singleton collection will be given by the unit of adjunction, that is $\forall x \in T$ $[x] = \eta_T(x)$.

The aggregation is defined as follows: for $f : X \to U(Y)$ we have

$$\mathrm{aggr}(f, \oplus) = f^{\#}.$$

Since $f^{\#} \in Hom\mathcal{A}$ we obtain

$$f^{\#}(x{+}{+}y) = f^{\#}(x) \oplus f^{\#}(y)$$

that is (2).

The commutativity of diagram (6) implies for $x \in X$

$$(U(f^{\#}) \circ \eta_X)(x) = U(f^{\#})([x]) = f^{\#}([x]) = f(x)$$

that is (1).

Example 3.2. If $\mathcal{A}$ is the category of sets with one binary operation (subject to no restrictions) and the arrows are such binary algebra homomorphisms we obtain binary labelled trees.

An ordered binary rooted tree (OBRT) is a binary rooted tree which has an additional linear order structure (referred to as left/right) on each set of siblings. An X-labelled OBRT (LOBRT/X) is an OBRT together with a function from the set of terminal nodes to X. A free algebra $F(X)$ expression has the usual representation as a binary tree. For example $(xy)z$ is represented in figure 1.

The concatenation of two trees t_1 and t_2 (or of two expressions) is the binary tree

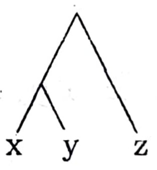

FIGURE 1. Expression

having t_1 as left subtree and t_2 as right subtree. □

If $\mathcal{A}$ is a subcategory of the category **Mon** of monoids we get an analogous of Theorem 1 for strong collections.

Theorem 3.3. *If $\mathcal{A}$ is a subcategory of* **Mon**, *$U : \mathcal{A} \to \mathcal{T}$ is the forgetful functor and F is his left adjoint, then the adjunction $< F, U, \eta, \varepsilon >$ determines a strong data collection.*

Proof. It is a variant of theorem 3.1 proof. The unique morphism $f^{\#}$ from diagram (6) is a monoid-homomorphism and for $T \in Ob\mathcal{T}$, $F(T)$ is the free monoid over T. Since $(F(T), +\!\!+, [\,])$ is a monoid we obtain (3) and (5); the commutativity of (6) implies (2), and $f^{\#}$ monoid-homomorphism implies (1) and (4).

Theorem 3.4. *If* **Bu** *is the category of algebras with one binary operation and neutral element, $U :$ **Bu** $\to \mathcal{T}$ is the forgetful functor and F is his left adjoint, then the adjunction $< F, U, \eta, \varepsilon >$ determines a zero-element data collection.*

The proof is analogous to that of theorem 3.3.

Example 3.5. Let add to collections from example 3.2 the empty tree (having no node). If z is the empty tree, the concatenation is extended as follows:

$$z{+\!+}t = t{+\!+}z = t$$

for each tree t.

Let $(B, \oplus, u)$ be an algebra with a binary operation $\oplus$, neutral element u and $f : A \to B$. The aggregation is extended as follows

$$\mathrm{aggr}(f, \oplus)(z) = u.$$

4. Algebraic properties of aggregation

Theorem 4.1. *If $f : X \to U(Y)$ then the following identity holds*

$$\mathrm{aggr}(f, \oplus) = \varepsilon_Y \circ F(f) \tag{7}$$

Proof. Applying F to (6) we have

$$\begin{tikzcd} F(X) \arrow[r, "F(\eta_X)"] \arrow[dr, "F(f)"'] & FUF(X) \arrow[d, dashed, "FU(f^{\#})"] \\ & FU(Y) \end{tikzcd} \tag{8}$$

The naturality of ε gives us

$$\begin{tikzcd} FUF(X) \arrow[r, "FU(f^{\#})"] \arrow[d, "\varepsilon_{F(X)}"'] & FU(Y) \arrow[d, "\varepsilon_Y"] \\ F(X) \arrow[r, "f^{\#}"'] & Y \end{tikzcd} \tag{9}$$

that is,

$$\begin{aligned} \varepsilon_Y \circ F(f) &= \varepsilon_Y \circ FU(f^{\#}) \circ F(\eta_X) && \text{(from(8))} \\ &= f^{\#} \circ \varepsilon_{F(X)} \circ F(\eta_X) && \text{(from(9))} \\ &= f^{\#} && (\epsilon_F \circ F_\eta = id) \ \square \end{aligned}$$

Remark 4.2. Theorem 4.1 also holds for strong collections.

Proposition 4.3. The following holds

$$\mathrm{aggr}(\eta, ++) = id_{F(.)} \tag{10}$$

Proof. From theorem 4.1 and adjunction we have

$$\mathrm{aggr}(\eta, ++) = \varepsilon_{F(T)} \circ F(\eta_T) = id_{F(T)}. \square$$

Remark 4.4. The identity (10) also holds for strong collections.

Proposition 4.5. *If $h \in Hom_A(T,S)$ where $T = (T, \oplus)$ and $S = (S, \otimes)$ are algebras with one binary operation, then for each $f : A \to U(T)$ we have*

$$h \circ aggr(f, \oplus) = aggr(h \circ f, \oplus) \tag{11}$$

Proof. From theorem 4.1 we have

$$\begin{aligned} \mathrm{aggr}(f, \oplus) &= \varepsilon_T \circ F(f) \\ \mathrm{aggr}(Uh \circ f, \otimes) &= \varepsilon_S \circ F(Uh \circ f) \end{aligned}$$

and

$$\begin{aligned} h \circ \varepsilon_T \circ F(f) &= \varepsilon_S \circ FU(h) \circ F(f) && (\text{ naturality of } \varepsilon) \\ &= \varepsilon_S \circ F(U(h) \circ f) && (F \text{ functor}) \\ &= \mathrm{aggr}(Uh \circ f, \otimes) && (12) \\ &= \mathrm{aggr}(Uh \circ f, \otimes), \end{aligned}$$

since $Uh = h$. $\square$.

Remark 4.6. Formula 11 also holds for zero-element and strong data collections.

Proposition 4.7. If $g : A \to B$ and $f : B \to C$ where $(C, \oplus)$ is an algebra with one binary operation then

$$\mathrm{aggr}(f, \oplus) \circ F(g) = \mathrm{aggr}(f \circ g, \oplus) \tag{13}$$

Proof. From theorem 4.1, because F is a functor we have

$$\begin{aligned} \mathrm{aggr}(f, \oplus) \circ F(g) &= \varepsilon_C \circ F(f) \circ F(g) \\ &= \varepsilon_C \circ F(f \circ g) \\ &= \mathrm{aggr}(f \circ g, \oplus). \square \end{aligned}$$

5. The importance of aggregation

The aggregation operator first appeard in John Backus' article [3]. In [4] is called *pump*; in [5] *aggr*. In [18] appears as *fold* and is the basic operator for database query and optimization. Its importance is that aggregation axioms and identities (7), (10), (11), (13) can be used for query optimization purpose. Also, the usual operators of relational algebra ([17, 16, 1]) can be expressed by aggregation as follows.

P1. Collapse. If we have a data collection type

$$C = (\tau',\ [],\ [x],\ +\!+,\ \mathbf{aggr})$$

where τ' is a collection of τ-type collections then the collapsing can be expressed as

$$\text{collapse}(x) = \text{aggr}(id, +\!+)(x).$$

P2. Selection. If p is a predicate, we have for strong collections

$$\sigma_p(x) = \text{aggr}(f_p, +\!+)$$

where $f_p = \lambda x.\text{if } p(x) \text{ then } [x] \text{ else } []$.

P3. Map or apply-to-all.

$$\text{map}(f, x) = \text{aggr}(f, ++)(x).$$

Remark 5.1. Map is intimately related to the free functor and adjunction. In fact we may write

$$\text{map}(f, .) = F(f).$$

P4. Project.

$$\begin{aligned}\text{project}_{a_1,\dots,a_n}(x) &= \text{map}(\lambda\{a_1 = x_1, \dots, a_n = x_n, \dots\}. \\ &\quad \{a_1 = x_1, \dots, a_n = x_n\}, x).\end{aligned}$$

P5. Join. We introduce

$$\text{filtermap}(p, f, x) = \sigma_p(\text{map}(f, x))$$

We have

$$\text{join}(xs, ys, f, g, h) = \text{map}(\psi, xs)$$

where

$$\psi = \lambda x.\text{filtermap}((\lambda y.f(x) = g(y)), h(x,y), ys).$$

P6. Powerset. If $C(T)$ is the set-type collection over T, then the set of all sets over T (T finite) can be expressed by:

$$\begin{aligned}
&\text{power} : C(T) \to C(C(T))\\
&\text{power}([\,]) = [[\,]]\\
&\text{power}(\text{add}(y, X)) = \text{power}(X) {+\!+} \text{map}(h)(\text{power}(X))
\end{aligned}$$

where $h(c) := \text{add}(y, C)$ and $\text{add}(y, C) = [y]{+\!+}C,\ y \in T,\ c \in C(T)$.
The generalization of this construction for other collection types is easy.

Other operations. The following operations may be expressed through aggregation:

- set membership

$$e \in X = \text{aggr}(f, \vee)(x) \qquad \text{where } f = \lambda z.(z = e);$$

- set difference

$$x \backslash y = \text{aggr}(f, \cup)(x) \qquad \text{where } f = \lambda z.\text{if } \neg(z \in y) \text{ then } [z] \text{ else } [\,];$$

- set intersection

$$x \backslash y = \text{aggr}(f, \cup)(x) \qquad \text{where } f = \lambda z.\text{if } (z \in y) \text{ then } [z] \text{ else } [\,];$$

- quantifiers

$$\begin{aligned}
(\forall x.p(x))(y) &= \text{aggr}(f, \wedge)(y)\\
(\exists x.p(x))(y) &= \text{aggr}(f, \vee)(y) \qquad \text{where } f = \lambda z.p(z).
\end{aligned}$$

6. Applications

We can find again classical data collection types taking various target categories for F functor.

Example 6.1. Let $\mathcal{A}$ be the categories of algebras with one binary operation. We obtain the data collection type from example 3.2. □

The next examples are on strong data collections.

Example 6.2. If $\mathcal{A}$ is the category of algebras with one binary operation having neutral element we obtain the data collection type from example 3.5. □

Example 6.3. If $\mathcal{A} =$ **Mon**, where **Mon** is the category of monoids we obtain list data collection types.

If $M \in Ob\mathcal{T}$, then $F(M)$ is the free-monoid generated by M (the set of lists having M-type elements), and for $f \in Hom\mathcal{T}$, $F(f)$ applies f to each element of the list (*apply-to-all* or LISP *mapcar* function). In fact F is the free-functor, and his right adjoint U is the forgetful functor.

If x is the list $[x_1, \ldots, x_n]$ and $f : X \to Y$, where $(Y, \oplus)$ is a monoid, then

$$\mathrm{aggr}(f, \oplus)(x) = f(x_1) \oplus \ldots \oplus f(x_n).$$

The concatenation is the usual list concatenation operation, the empty collection is the empty list, and the single-element collection is the one-element list. □

Example 6.4. If $\mathcal{A} =$ **CMon**, where **CMon** is the category of commutative monoids, we obtain the multiset(bag) data collection types. The empty collection is the empty multiset, and the concatenation is the multiset union (that is, the number of occurrences of an element is the number of occurrences in the first multiset plus the number of occurrences in the second).

If x is the multiset $[x_1, \ldots, x_n]$ with X-type elements and $f : X \to Y$, where $(Y, \oplus)$ is a commutative monoid, then

$$\mathrm{aggr}(f, \oplus)(x) = f(x_1) \oplus \ldots \oplus f(x_n). \ \square$$

Example 6.5. If $\mathcal{A}$ is the category **CUSL** of upper complete semilattices we obtain set data collection types. The empty collection is the empty set, the singleton collections are singleton sets, and the concatenation is the usual union.

If $\{x_1, \ldots, x_n\}$ is a set with X-type elements, $f : X \to Y$, where $(Y, \vee)$ is an upper complete semilattice then

$$\mathrm{aggr}(f, \vee)(x) = f(x_1) \vee \ldots \vee f(x_n) \ \square$$

Example 6.6. If $\mathcal{A}$ is the category of monoids with one binary idempotent operation, then we obtain *oset* type data collections (lists without duplicates). The empty and singleton collection are the same as for ordinary lists, but the definition of concatenation is $x +\!\!+ y = x \cup (y - x)$ where $y - x$ is the list of elements in y, but not in x. □

Example 6.7. If $\mathcal{A}$ is the category of ordered idempotent monoids we obtain sorted collections. The concatenation is list merging. A variant of these collection type parametrized by a function f whose range has a partial order "$\leq$" given by $x \leq y \Longleftrightarrow f(x) \leq f(y)$, and called *sorted*[*f*] appear in [10] and [11]. □

Remark 6.8. Some classical aggregation operations for databases, such as count or sum are exception from the frame of example 6.2. They could be defined as $\text{count}(x) = \text{aggr}(f, +)$, where $f(y) = 1$, and $\text{sum}(x) = \text{aggr}(id, +)$.

We have

$$\text{count}(\{2\}) = 1 = \text{count}(\{2\} \cup \{2\}) \neq \text{count}(\{2\}) + count(\{2\}) = 2$$

$$\text{sum}(\{2\}) = 2 = \text{sum}(\{2\} \cup \{2\}) \neq \text{sum}(\{2\}) + \text{sum}(\{2\}) = 4$$

For these cases f is not a homomorphism of complete upper semilattices. □

There is another way to define aggregation operators, having theorem 4.1 as starting point. If $f : T \to S$, $\oplus : S \times S \to S$ is a binary operation and $[x_1, \ldots, x_n] \in F(T) = \tau$ we define

$$\varepsilon_S([x_1, \ldots, x_n]) = x_1 \oplus \ldots \oplus x_n.$$

The definition of aggregation operator is now

$$\text{aggr}(f, \oplus) : \tau \to S, \;\; \text{aggr}(f, \oplus) = \varepsilon_S \circ F(f)$$

We shall call this operator generalized aggregation operator. In fact, this is the classical aggregation operator as defined in Functional Programming. This operator will have the property (2) or (4) only if $f : T \to S$ is a homomorphism of appropriate algebras.

Alternatively, the aggregation operator can be seen as a mapping from a colection type $< \tau, [x], ++, \text{aggr} >$ to a binary operation algebra $(S, \oplus)$. It will have the desired properties if the category which has the algebra $(S, \oplus)$ as object is a subcategory of the category corresponding to the collection type. This property is called *well-definedness* and it is studied in [9, 10, 11, 6], but the way of expression is not categorical. For sum and count, $(\mathbb{N}, +)$ is not an upper complete semilattice, but a commutative monoid. Thus sum and count will be well-defined for lists and multisets and ill-defined for sets. Adjunction leads to well-definedness.

References

[1] S. Abiteboul, R. Hull, V. Vianu – *Foundations of Databases*, Draft book, 1994.

[2] M. P. Atkinson, P. W. Trinder, D. A. Watt - Bulk Type Constructor, *Technical Report*, FIDE/93/61, 1993.

[3] John Backus – Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *CACM* 21(8), pp. 613-641, 1978.

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

[5] Catriel Beeri, Tova Milo – Functional and predicative programming in OODB's, *PODS 92*, 1992.

[6] P. Buneman, S. Naqvi, V. Tannen, L. Wong – Principles of programming with complex objects and collectin types, *TCS* no. 1, pp. 1-46, 1995.

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

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

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

[10] 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.

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

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

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

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

[15] J. W. Schmidt – Some high level language constructs for data of type relation. *ACM Transactions on Database Systems* 2(3), pp. 247-261, 1977.

[16] J. D. Ullman – *Principles of Database Systems*, Computer Science Press, 1982.

[17] J. D. Ullman – *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1988.

[18] Bennet Vance – Towards an Object-Oriented Query Algebra, *Technical Report* CS/E 91-008, Oregon Graduate Institute, 1991.

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

E-mail address: `tradu@cs.ubbcluj.ro`