Make More of Data Types

Heinrich C. Mayr

Institut für Informatik II
University of Karlsruhe
Western Germany

Introduction

This position paper refers to some observations I made at the 'High Level Abstraction Workshop', and it exhibits a way to attack abstraction and specification problems in the database field. Because of the fixed page limit this will be done in a more or less cursory collection of remarks.

The Problem

There is a wide-spread feeling that, beyond technical and implementation-oriented work, the database field still suffers from a rather nonuniform and unprecise terminology caused by the lack of a fundamental database theory. This deficiency may be one of the reasons why database people have difficulties in making their problems and issues clear to representatives of other, more consolidated fields. The Pingree Park workshop confirmed me in that impression.

E.g., in databases we are using circumscriptions like

> 'a type is a precise characterization of some structural and behavioural properties common to a set of objects'

as definitions although their constituents are mostly undefined. So we need not be surprised if this kind of ambiguity leads us often to give the same concept different names in order to catch the intended semantics as is done, e.g., in the case of 'modelling concepts', 'abstraction forms', 'relationship types'.

On the other hand, we try to adopt notions and techniques from other disciplines but do not do so consequently. One of the most distinct examples for this fact is the use of types. Types have a sound mathematical foundation in the work of the ADJ-group [2,4,5,6,16] and are commonly accepted to be advantageous for many purposes. However, their full power is not yet exploited in databases. Therefore, I will concentrate in this paper on types, derive formal notions of 'data types' and 'abstract data types', and point out some of the advantages they have for the database field.

Note that I do not intend to run down the conceptual frameworks elaborated up to now, the more so as formal rigour does not always fit into the world of practical needs. I only think that time has come to bolster this framework by more and precise fundamentals.

Types and Data Types

For the remainder, two (simplified and, by nature, informal) premises are substantial:

Premise 1:

> Common to all disciplines dealing with information processing is the manipulation of objects, where
> - the objects, called data, represent models (in the sense of 'human cognitive structures', i.e. mental things [3]) of some part of the - as such accepted - real world.
> - the manipulations, called operations, represent processes that have taken place, will take place or might take place in the real world or in the mental world of objects.

Premise 2:

> Neither parts of the real world nor their representing data exist by themselves. Their existence is due to the generation by processes and operations, respectively.

Accepting theses premises there is an immediate consequence:

> 'Databases', 'Knowledge bases' and 'Data spaces' of program systems have no fundamental differences. They differ at most in the concepts and techniques, models are represented. Thus, what we need, is to get a better grip on these concepts and techniques.

It is a formal notion of data type drawn from the type definition in [6,7,14] that may help us to achieve this goal (if, without excluding the possibility of later extension, we assume operations to be functions):

A type T is an algebra $T=(\Sigma,\Omega)$ of a set Σ of sets, called carriers, and a set Ω of functions amongst the carriers.

A data type DT is a triple $DT=(T,S,O)$ where

(a) $T=(\Sigma,\Omega)$ is a type,
(b) $S \in \Sigma$ is a distinguished carrier, called the value set of DT,
(c) $O \subseteq \Omega$ is a distinguished set of functions, called type operations, having S in their

domain or range list,

(d) $\Sigma \setminus \{S\}$ is a set of value sets of other data types,

(e) for each $v \in S$ there is a constant operation $v: \rightarrow S$ and $v() = v$.

The elements of S are called values or instances of data type DT. They may be used to represent models.

(d) completes the data type definition in the sense that if a set R is a carrier of data type DT then R itself is the value set of a data type. More laxely, data types may only be built up from data types.

Evidently now, the question arises how data types may be specified and presented with respect to the fact that O and S may be infinite.

To attack this question it is convenient to restrict oneself to data types having a finite number of constants that cannot be generated by non-constant operations. For, only such data types are of interest in the finite world of computer assisted information processing. They mostly provide operations that 'construct' type values starting from a strongly limited number of initial constants like, e.g., the empty set, the empty database etc. In these cases we may associate to each type value one or more operational expressions of the form:

Expression	is associated to	Value
$v()$		v
$o_1(o_2(...),..,o_n(...))$		value resulting from calculating ('performing') the expression

Thus, on a more abstract level, we need not consider S explicitly when dealing with the type operations and the word algebra established by these operations. It is convenient to speak, at this level, of 'abstract types' and 'abstract data types' respectively.

Introducing these concepts formally in this paper would mean to strike out other important aspects. So I only indicate that the definitions are formulated analogously to those of type and data type using the framework of [6]. Note, however, that both, abstract data types and data types abstract from any kind of 'implementation'. They differ only in that data types account also for some common representation of type values. This view is somewhat different from other conceptions but it reflects the fact that, in practice, we do not worry about implementation when using or speaking of the data types of a certain system or module interface.

On the other hand, this view does not impede us in the desired distinction between, e.g., data types 'Arabic integer numbers' and 'Roman integer numbers' but allows us to associate with them the same abstract data type.

The operations of an abstract data type establish a congruency relation C in the word algebra of operator expressions. Thus, our initial question of presenting data types is reduced (apart from syntactics) to the specification of C and to the association of semiotic signs to expressions.

A variety of techniques for the former purpose has been proposed so far [e.g.,2,5,7,8,10,12,13,14,15] and the future will show which of them will survive in practice.

The type framework receives its full power by the concept of parameterization [16] leading to the socalled 'parameterized types' ('type constructors' 'type concepts'). Adjusted to (abstract) data types, parameterization allows the specification of classes of (abstract) data types with different carriers and even partly different operation semantics. Consider, e.g., a parameterized (abstract) data type array[CARRIER,INDEX] that lets open which and at which position elements may be entered into an array. By determining these parameters an (abstract) data type is gained.

Data Types in the Database Field

A number of advantages like encapsulation, abstraction implementation independence etc. have been stated concerning the use of types. The programming language area tries to exploit these advantages more and more, e.g., by developping adequate design and programming languages [9,10,15,18].

Against that, in databases there is a trend to doubt types being powerful enough to capture all important aspects. Cited examples are, among others

- constraints
- complexity of structures and operations
- live-span of data
- reliability in the case of harware malfunction
- exception handling
- concurrency.

Consequently, one asks for extensions improving the 'semantic capabilities' of types. Given the state of the art in type specification and implemented type support, this demand is legitimate from a practical point of view. Theoretically, however, it starts on a wrong assumption since - provided that premises 1 and 2 are accepted - data types as introduced before have the necessary capabilities. E.g., alle the listed 'counterexamples' are expressible in terms of data and operations and thus may be captured if appropriate data type specification techniques are supplied.

What I want to propose is, therefore, to start from data type as the basic and overall concept and then

- to derive formal and unambigous notions for particular aspects and to develop techniques for their specification,
- to determine the 'typical' (!) differences between databases and other research areas,
- to identify a set of fundamental (parameterized) data types that may be used as a constructive [17]terminological basis.

A first step into this direction is made by viewing databases as type values themselves, an approach that has been sketched in [2,11]. A variety of straightforward questions then arise:

- Data models become normal parameterized abstract data types and there is 'only' the question of how to specify them.

- Database schemes become normal parameter assignments and there is the question of what syntactical means are needed for their formulation.
- Constraints become normal operation properties and there is the question of how to formulate the operation definitions comprehensibly and completely. Different kinds of constraints concerning interrelationships between different 'lower' types (like, e.g., sets, areas and records in the DBTG network model) may then be identified in a clear an straightforward manner [11].

Another advantage of referring to a precise type framework is the fact that hitherto weak concepts may be related and may be defined and studied more formally. E.g., if we interprete 'conceptual modelling' as the execution of intellectual model building and relating processes (generalization, aggregation, classification etc.), then these processes have counterparts on the type level in the form of morphisms amongst data types. These morphisms do not exceed the framework mentioned so far, since they may be treated as operations of a data type whose values are datatypes themselves. This conception is already indicated in [1].

There have been voices criticizing the strong association of operations with single types. I may not follow these reproaches since just this sharp delimitation is one of the big advantages of the type framework and it does not prevent us from expressing certain semantics by giving operations of different types the same names (e.g., '+' for number and string types). For, if we want to do so, we should and must specify a data type that integrates the intended ones. So we get a uniform treatment of semantics.

Data Type Specification

The main obstacle in using types for database purposes consists in the actual lack of universal, formal, comprehensible and practice-oriented type specification techniques. A variety of approaches have been discussed in the literature but none of them became widely accepted or used up to now.

Comprehensible techniques having a wide range of applicability mostly are not formal enough in order to allow for correctness and completeness proofs - a demand that is made meanwhile in practice, too. Against that, formal techniques sometimes are not universal enough or lead to complex and unoverlookable specifications. Mainly the actual favorites, i.e., the algebraic technique and related approaches based on rewrite systems [14] seem to suffer from the purely recursive way of defining operations: A number of concepts that are algorithmically rather easy to manage become unpleasant to handle . Consider, e.g., the propagation of record deletion in DBTG network databases.

Thus, still a lot of work will have to be done in developping appropriate and perhaps special purpose oriented techniques that together form a 'tool-kit' for databases on the basis of a strongly type-oriented discipline. A step into this direction is presented in [13] by introducing a specification technique that tries to overcome a number of problems in a straightforward and practice-oriented fashion.

Literature

[1] Brodie,M.L.(1979): The Application of Data Types to Database Semantic Integrity. Techn. Report TR-833, Univ. of Maryland.

[2] Ehrig,H.;Kreowski,H.J.;Weber,H.(1978): Algebraic Specification Schemes for Database Systems. Proc 4th VLDB, Berlin 1978, pp. 427-44o.

[3] v.Förster,H.(1969): Analysis and Synthesis of Cognitive Processes. Rep. Biol. Comp. Lab., Univ. of Illinois, Urbana.

[4] Goguen,J.A.(1977): Abstract Errors for Abstract Data types. Semantics and Theory of Computation Report 6, UCLA.

[5] Goguen,J.A.(1977): Algebraic Specification.Sem. and Theory of Comp. Report 9, UCLA.

[6] Goguen,J.A.;Thatcher,J.W.;Wagner,E.G.;Wright,J. B.(1975): Abstract Data Types as Initial Algebras and Correctness of Data Representations. Proc. 'Comp. Graphics and Pattern Recognition and Data Structures',pp.89-93.

[7] Guttag,J.(1979): Notes on Type Abstraction. In Proc. 'Specification of Reliable Software', IEEE 79 CH 14o1-9c, pp.36-46.

[8] Hoare,C.A.R.(1972): Proof of Correctness of Data Representation. Acta Informatica 1, pp.271-281.

[9] Ichbiah,J. et al(1979): Preliminary Ada Reference Manual. SIGPLAN Notices, Vol 14, Nr6.

[10] Liskov,B.H.;Snyder,A.;Atkinson,R.;Shaffert,C. (1977): Abstraction Mechanisms in CLU. Comm. ACM 2o, Nr. 8, pp. 564-576.

[11] Lockemann,P.C.;Mayr,H.C.;Weil,W.H.;Wohlleber, W.H.(1979): Data Abstractions for Database Systems. ACM TODS, Vol 4, Nr. 1, pp.6o-75.

[12] Majster,M.E.(1977): Extended Directed Graphs, a Formalism for Structured Data and Data Structures. Acta Inform. 8, pp. 37-59.

[13] Mayr,H.C.;Lockemann,P.C.;Dittrich,K.R.(198o): Operational Replacement Schemes. Int. Report 11/80, Univ. of Karlsruhe.

[14] Musser,D.R.(1979): Abstract Data Type Specification in the AFFIRM System. Proc. 'Specif. of Reliable Software',IEEE 79 CH 14o1-9c.

[15] Silverberg,B.A.;Robinson,L.;Levitt,K.N.(1979): The HDM Handbook, Vol II. Report Aoo6, SRI International.

[16] Thatcher,J.W.;Wagner,E.G.;Wright,J.B.(1978): Data Type Specification: Parameterization and the Power of Specification Techniques. Proc. SIGACT 1oth Symp. on 'Theory of Computation', pp. 119-132.

[17] Wedekind,H.; Ortner,E.(1980): Systematisches Konstruieren von Datenbankanwendungen. Carl Hanser Verlag, München.

[18] Wulf,W.A.;London,R.L.;Shaw,M.(1976): An Introduction to the Construction and Verification of ALPHARD Programs. IEEE Trans. on Software Eng. SE-2.4, pp. 253-256.