Towards an Algebraic Semantics for Database Specification

H.-D. Ehrich, K. Drosten, M. Gogolla

Institut für Informatik, TU Braunschweig
Postfach 3329, D-3300 Braunschweig, Fed. Rep. of Germany

In the framework of a modal-algebraic approach to database semantics, the specification of abstract object types on the basis of abstract data types is studied. As a semantic framework for determining admissible states and state sequences, a standard universe of "possible objects" and their interrelationships has to be associated with a schema specification. This paper gives a construction of such a standard universe from a given key system including certain constraints. There is also an abstract algebraic characterization of the universe (up to isomorphism) in terms of final algebras. Within this framework, a general definition of admissible states and state sequences as the semantics of a schema specification is discussed briefly.

1. INTRODUCTION

Many approaches to software design and development agree that, before going into implementation, there should be a specification phase where the essential features of the system are to be described in an abstract and implementation independent way. Much agreement is still obtained when requiring that the specification method should be formal and should have a precise semantics. Less agreement, however, is obtained when discussing which is the right specification method for which purpose.

We share the basic assumption that logic provides the appropriate framework for studying specification methods, and model theory is the right basis for studying semantics.

For abstract data types, equational logic and their algebraic semantics have been discussed widely [GTW78, Eh82, EM85]. There are many results showing that a large variety of useful abstract data types can be characterized this way.
In principle, also database-oriented systems can be specified by this method, as shown by EKW78 and DMW82. We feel, however, that this is somewhat artificial and does not capture the intuition in an adequate way. In database-oriented systems, imperative concepts like states and updates play a central role, and this is obscured when modelling everything in purely applicative terms.

More natural is the use of modal logic as in GMS83 and KMS85, where states are modelled as possible worlds. In ELG84 and LEG85, we followed this basic approach.

There is, however, a semantical problem with this approach. Some quantifications run over all "actual" objects in a current state, and some quantifications extend over all "possible" objects that might be in some state. The problem is how such a universe of possible objects looks like or, more precisely, how a standard universe of possible objects can be associated with a database specification.

This paper suggests a solution to this problem. A standard universe of possible objects is constructed from given keys and certain constraints. The key system of a database is regarded as an extension of its data type system, adding object sorts and keys as functions from object sorts to data sorts or object sorts. This way, the object level is clearly separated from the data level of a database specification.

2. DATA AND OBJECT MODELLING

Designing a database is a pretty complex process and, therefore, it is desirable from a logical point of view and from an implementation point of view to partition the task into several manageable subprocesses. For this reason the specification of a conceptual database schema involves three layers, namely the data, object and transaction layer, and each layer is built upon the previous one.

```
Data
  
Objects
  
Transactions
```

On the data layer, a collection of abstract data types, for instance specified algebraically by one of the existing approaches
(e.g. universally quantified equations and initial algebra semantics), is introduced. The abstract data types have a fixed interpretation not changing in time. Thus, there is only one fixed world of data elements, which is part of every possible world constituting a database state.

On the object layer, there are objects as well as attributes for and relationships between them, determining the structure of a database state. Information about objects is retrieved via functions returning values of abstract types already specified on the data layer. Database states change in time, thus varying sets of actual objects chosen from a universe of possible objects have to be characterized and described, preferably in a temporal logic.

On the transaction layer, state-dependent operations possibly changing the database contents are defined, but aggregate functions belong to this layer as well. The transactions include simple insert, delete and update functions, and more complex ones can be constructed from these. Generally, transactions are specified independently of their implementation by means of pre/post conditions.

The above ideas have been explained in detail in a series of papers [EL84, Eh84, LEG85, Li85, LSE85, Eh86]. Here we will concentrate on the relationship between the first two layers and especially study the construction of the universe of possible objects relative to a fixed set of data types on the data layer and a fixed set of so called key functions on the object layer.

The first layer in our approach, where the data types are fixed, again can be structured. On the one hand there will be some standard types occurring in nearly every application and similar to common data types in programming languages. On the other hand there will be primitive application dependent types relying on the world to be modelled. Both kinds of types can be used in connection with type constructors to build complex data types rich enough to express all relevant information about object types. The semantics of the data layer is a fixed algebra.

On the object layer object sorts and object functions are introduced. The object functions can be classified into a group of so called keys, identifying an object uniquely, and additional object functions having no influence on the generation of objects, but which are necessary to express arbitrary attributes and relationships. The keys can be understood as constructors (in analogy to the theory of abstract data types) for the object
sorts. One can distinguish between data keys, giving for an object relevant data information, and object keys, yielding another object for a given one. The semantics of the object layer is a class of algebras, where the data part of each algebra is identical to the fixed data algebra. Each algebra represents a real world situation.

Example: The above ideas and the terminology are explained by the following small example representing a part of an information system involving companies and their employees. Sorts and functions are displayed in a so called signature graph, as introduced in [GTW78], but data sorts and object sorts as well as key and non key functions are displayed differently.

On the data layer we have the types Nat, Text and Date, all represented in the diagram by boxes. Nat and Text are assumed to be basic types, whereas Date can be understood as a record type consisting of primitive types Day, Month and Year. Additionally, we need appropriate functions, e.g. addition and multiplication for natural numbers, concatenations for texts and selectors for records, not shown in the diagram above.

For the object layer, the example introduces two object sorts COMPANY and EMPLOYEE, represented by circles. Key functions are visualized by arrows. Thus a company is identified by the company name, the company sales and the company manager, and an employee is identified by her/his name, her/his salary and the company she/he works for. cname and sales are data keys, manager is an object key. Moreover, we have functions not representing keys, which are shown by dotted lines and which give additional, possibly derived information about objects. The following lines can be understood as definitions of number-of-employees and superior.

\[
\text{number-of-employees}(c) = |(\exists\text{EMPLOYEE};\text{works-for}(c)=e)|
\]

\[
\text{superior}(e) = \text{manager}(\text{works-for}(e))
\]
The main topic of the paper is a method to construct the universe of possible objects, in the example for the sorts COMPANY and EMPLOYEE relative to the given six key functions.

3. DATA SPECIFICATIONS

To make our ideas on database specification precise we have to review some of the basic notions of the theory of abstract data types. The mathematical foundations of the field have first been developed in the pioneering paper GTW78. Since then, many papers studied concepts for the modularisation of algebraic specifications. Extensions of abstract data types play a dominant role and there are several meanings that can be associated with them: initial or free semantics [GTW78, Eh82], final semantics [Wa79, WPPDBB83, Ja85] and behavioural semantics [HR83]. Apart from this, parametrization of abstract data types is important for modularizing specifications [EKTWWB81, Eh82].

A signature \( \Sigma=(S, \Omega) \) consists of a set \( S \) of sorts and an \( S^2 \times S \)-indexed family \( \Omega \) of operation symbols, which are denoted in the usual way \( \omega:s_1 x...x s_n \rightarrow s \). A \( \Sigma \)-algebra \( A \) consists of carrier sets \( s_A \) for each \( s \in S \) and functions \( \omega_A:s_1 A \times ... \times s_n A \rightarrow s_A \) for each \( \omega:s_1 x...x s_n \rightarrow s \) in \( \Omega \). A \( \Sigma \)-algebra \( B \) is a subalgebra of the \( \Sigma \)-algebra \( A \) (denoted by \( B \subseteq A \)), if \( s_B \subseteq s_A \) holds for all \( s \in S \), \( \omega_B \) is the restriction of \( \omega_A \) for all \( \omega \in \Omega \), and \( B \) is closed under all functions. A \( \Sigma \)-algebra morphism \( h:A \rightarrow B \) is an \( S \)-indexed family of mappings \( h_s:s_A \rightarrow s_B \) that respects the operations. In the category \( \Sigma \text{-alg} \) of all \( \Sigma \)-algebras and \( \Sigma \)-algebra morphisms there exists an initial object, i.e. an algebra from which there is a unique morphism to any other algebra, and a final object, i.e. an algebra such that there is a unique morphism from any other algebra to it. The initial algebra is isomorphic to the term algebra, and the final one is (in the absence of axioms) of no interest, because it has singleton sets for all sorts.

A specification \( D=(\Sigma, E) \) consists of a signature \( \Sigma \) and a set \( E \) of \( \Sigma \)-formulas as axioms. If the axioms \( E \) consist only of universally quantified equations, then the full subcategory \( D\text{-alg}\Sigma \text{-alg} \) of all \( \Sigma \)-algebras satisfying the axioms has an initial algebra, isomorphic to the quotient of the term algebra by the least congruence generated by the equations. But also more general forms of axioms have been studied [WPPDBB83, Go85]. If the initial and final algebra are isomorphic, the data type is called monomorphic.
A signature morphism $f: \Sigma_1 \to \Sigma_2$ is a mapping from sorts to sorts and operators to operators such that the image of $\omega_1 x_1 \ldots x_n \to s$ works on the image of the sorts, i.e., $f(\omega): f(s_1) x\ldots x f(s_n) \to f(s)$ is part of $\Sigma_2$. A signature morphism induces a forgetful functor $\mathcal{F}: \Sigma_2\text{-alg} \to \Sigma_1\text{-alg}$ by sending a $\Sigma_2$-algebra $A$ to $\mathcal{F}(A)$ where $s\mathcal{F}(A) = f(s)A$ and $\omega\mathcal{F}(A) = f(\omega)A$ and by sending a $\Sigma_2$-algebra morphism $h: A \to B$ to $\mathcal{F}(h): \mathcal{F}(A) \to \mathcal{F}(B)$, where $\mathcal{F}(h)_s = h f(s)$ holds. $\mathcal{F}(A)$ and $\mathcal{F}(h)$ are called the $\Sigma_1$-reducts of $A$ and $h$, respectively (denoted by $A_1|_1$ and $h|_1$). A specification morphism $f: \Delta_1 \to \Delta_2$ is a signature morphism $f: \Sigma_1 \to \Sigma_2$ such that $\forall(D_2\text{-alg}) \in \Delta_1\text{-alg}$.

An extension is in general a translation from $\Sigma_1$-algebras to $\Sigma_2$-algebras, reversing the direction of $\mathcal{F}$. The standard case of an extension is specified by an inclusion $f: \Delta_1 \to \Delta_2$, which is a specification morphism. This models the process of adding new functions and types to an existing type structure. All notions of extensions studied in the framework of abstract data types are constructive in the sense that the extended type is based on the free extension [Eh82]. In this respect, key extensions to be introduced in the next section are different.

Another important structuring method are parametric specifications. A parametric specification consists of a parameter specification $P = (\Sigma_P, E_P)$ and a body specification $B = (\Sigma_B, E_B)$, such that $\Sigma_P \subseteq \Sigma_B$ and $E_P \subseteq E_B$ holds. A parametric specification induces a functor $f^*: P\text{-alg} \to B\text{-alg}$, the free construction [EKTW81], which can be understood as a type constructor yielding a $B$-algebra for each parameter algebra. It is required that the construction is persistent, i.e., $\mathcal{F}(f^*(A))$ should be isomorphic to $A$ in a natural way for all parameters $A$. Parameter substitution can be made precise by means of pushouts [EKTW81, Eh82].

For what follows, we assume a monomorphic abstract data type DATA to be given. DATA can be constructed by extensions of basic abstract types and applications of parametric specifications to already generated types. Let $\Sigma_D = (\Sigma_D, E_D)$ be its signature, from now on called the data signature, and let $E_D$ be its axioms, which together with $\Sigma_D$ constitute the data specification $(\Sigma_D, E_D)$. The results are based on the following assumptions about DATA.

1. All carriers $s_{DATA}, s_{DATA}'$ are nonempty.
2. DATA does not have proper automorphisms, i.e. the only one is identity.

These assumptions do not seem to be too restrictive. Empty carriers are of no practical relevance. Moreover, most practical data types are minimal algebras (for instance, initial algebras are minimal), and these never have proper automorphisms.
4. KEY SPECIFICATIONS

Specifying the object level on top of a given data level may become a very complex task so that it is best done in subsequent steps. The first step consists of identifying all objects which may possibly be met in a database application. For this purpose, we propose a key concept which is similar to that one in the relational data model. It is based on the idea that any possible object (of type $s \in S_0$) can be uniquely described by data values which are observable under its (composite) keys.

For what follows, let a data signature $\Sigma_D = (S_D, \Omega_D)$ with a fixed interpretation DATA be given. To specify all possible objects in a database application it is necessary to provide
- the complete set $S_0$ of object sorts
- an $S_0 \times S$-indexed set family $\mathcal{K}$ of keys where $S = S_0 \cup S_D$.
Assuming that $S_0 \cap S_0 = \emptyset$ and $\Omega_D \cap \Omega_K = \emptyset$ the resulting signature $\Sigma_K = \Sigma_D + (S_0, \Omega_K)$ is called key signature.

To be more suggestive, a key $k \in K; s, t (s \in S_0, t \in S)$ is often denoted by $k: s \rightarrow t$. This is in accordance with the intended interpretation that $k$ is a unary function from objects of type $s$ to objects or data of type $t$. If $t$ is a data sort, $k$ is called data key, and it is called object key otherwise.

Example: Referring to the person/company example in section 2, the signature graph is obtained by forgetting about the dotted lines for non-key functions.

A key signature is meant to fix the universe of possible objects which may be dealt with in a database application. To model the idea that each object is characterized by its "observations", i.e. by its data values under (composite) keys, the notion of key sequence is introduced. The set of possible objects of type $s \in S_0$ is then given by all legal assignments of data values to key sequences.
Let \( s, t \in S \). A key sequence from \( s \) to \( t \) is defined recursively as follows:

(i) the empty sequence \( \varepsilon \) is a key sequence from \( s \) to \( s \)

(ii) if \( x \) is a key sequence from \( u \) to \( t \) and \( k : s \to u \) is a key, then \( kx \) is a key sequence from \( s \) to \( t \)

(iii) nothing else is a key sequence

Considering key signature graphs, key sequences from \( s \) to \( t \) correspond to directed paths from node \( s \) to node \( t \). The set of all key sequences from \( s \) to \( t \) is denoted \( L_{s,t} \). To construct the universe of possible objects, we are mainly interested in key sequences which end in data sorts. Accordingly, we define

\[
L_s = \bigcup_{r \in S_D} L_{s,r}
\]

Note that, for any \( r \in S_D \), \( L_r = L_{r,r} = \{ \varepsilon \} \).

Example: Referring to the employee/company example, we have

\[
L_{\text{EMPLOYEE}} = \text{(works-for manager)}^* \text{ salary}
\]

\[
\quad \text{v (works-for manager)}^* \text{ ename}
\]

\[
\quad \text{v works-for(manager works-for)}^* \text{ sales}
\]

\[
\quad \text{v works-for(manager works-for)}^* \text{ cname}
\]

\[
L_{\text{COMPANY}} = \text{(manager works-for)}^* \text{ sales}
\]

\[
\quad \text{v (manager works-for)}^* \text{ cname}
\]

\[
\quad \text{v manager(work-for manager)}^* \text{ salary}
\]

\[
\quad \text{v manager(work-for manager)}^* \text{ ename}
\]

Let a key signature \( \Sigma_K = \Sigma_D^+(S_0, \Omega_K) \) be given. Our universe \( \text{UNIV}(\Sigma_K) \) is constructed as a particular \( \Sigma_K \)-algebra for which the \( \Sigma_D \)-reduct is \( \text{DATA} \) (up to isomorphism). The latter is in accordance with the intention that once the data layer is specified, it should not be affected by later steps in database specification.

The construction of the universe \( \text{UNIV}(\Sigma_K) \) — for convenience, we often write \( U \) for \( \text{UNIV}(\Sigma_K) \) — is as follows.

The carrier set of sort \( s \in S_D \cup S_0 \) is given by

\[
s_U = \{ \psi | \psi : L_s \to \bigcup_{r \in S_D} r_{\text{DATA}} \text{ such that, for each } r \in S_D, \psi(L_{s,r}) \in r_{\text{DATA}} \}
\]

Since \( r_{\text{DATA}} \) is assumed to be non-empty for any \( s \in S_D \), \( s_U \) is non-empty. Even if there is no key sequence starting from \( s \), \( s_U \) contains one element, the empty mapping.

1) Whenever we speak of a \( \Sigma_D \)-reduct we refer to the image of the forgetful functor induced by the inclusion \( \Sigma_D \hookrightarrow \Sigma_K \).
For each data sort \( r \in \mathcal{S}_D \), \( L=r(\epsilon) \) so that \( r_\mathcal{U} = \{(\epsilon, a) \mid a \in r_\mathcal{DATA}\} \). There is an obvious 1-1-correspondence \( h: r_\mathcal{DATA} \rightarrow r_\mathcal{U} \) given by \( h(a) = (\epsilon, a) \) (actually \( \{(\epsilon, a)\} \) but we omit the set braces).

The operational structure of \( \mathcal{U} \) is given as follows. For data operators \( \omega: r_1 \times \ldots \times r_n \rightarrow r(r_1, \ldots, r_n ; i=1, \ldots, n) \) the corresponding function \( \omega_\mathcal{U}: r_1_\mathcal{DATA} \times \ldots \times r_n_\mathcal{DATA} \rightarrow r_\mathcal{DATA} \) is defined by

\[
\omega_\mathcal{U}((\epsilon, a_1), \ldots, (\epsilon, a_n)) = (\epsilon, \omega_\mathcal{DATA}(a_1, \ldots, a_n))
\]

From this, \( h \) gives an isomorphism from \( \mathcal{DATA} \) to the \( \Sigma_D \)-reduct of \( \mathcal{U} \).

For keys \( k: s \rightarrow t \), \( k_\mathcal{U}(\psi) = \lambda x \psi(kx) \)

where \( x \) ranges over \( L_t \). Informally speaking, each mapping \( \psi \in \mathcal{s}_\mathcal{U} \) is sent to the mapping \( \psi' \in \mathcal{U} \) satisfying \( \psi'(x) = \psi(kx) \) for each \( x \in L_t \). Note that \( k_\mathcal{U} \) is well-defined due to the fact that, for any key sequence \( x \) in \( L_t \), there is a key sequence \( kx \) in \( L_s \). If \( t \in \mathcal{S}_D \), we get \( k_\mathcal{U}(\psi) = (\epsilon, \psi(k)) \) where \( \psi(k) \in \mathcal{DATA} \), since \( L_t = \{\epsilon\} \).

Please verify that the construction of the carrier sets in \( \text{UNIV}(\Sigma_\mathcal{K}) \) is really based on the idea to characterize objects uniquely by their data observable under (composite) keys. For all objects of any type \( s \in \mathcal{S}_0 \), a uniform format of description is used; it is determined by the set of key sequences from \( s \) to data sorts. Referring to our introductory example, employees are described by means of the following pattern.

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>salary</th>
<th>ename</th>
<th>works-for sales</th>
<th>works-for cname</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nat</td>
<td>Text</td>
<td>Nat</td>
<td>Text</td>
</tr>
<tr>
<td></td>
<td>Nat</td>
<td>Text</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The set of possible employees contains all instances of the above pattern. In \( \text{UNIV}(\Sigma_\mathcal{K}) \), instances are represented by functions. Describing them by a cartesian product would not be an appropriate choice because there may be an infinite number of key sequences (cf. \( \text{EMPLOYEE} \)). Whenever there is a key \( k: s \rightarrow t \) (\( s, t \in \mathcal{S}_0 \)), all observations for objects of type \( t \) are also possible for objects of type \( s \). The mapping \( k_\mathcal{U} \) sends objects in \( s_\mathcal{U} \) to objects in \( t_\mathcal{U} \) in such a way that a coincidence of observations is guaranteed, at least partially.
Consider the special case that $\Sigma_K$ has data keys only. Let $k_i : s \rightarrow r_i$ ($r_i \in S_0; i = 1, \ldots, n; n > 0$) be all keys for $s \in S_0$. Since $L_s$ is finite, there is an obvious 1-1-correspondence from $s$ to the cartesian product $r_1, DATA \times \cdots \times r_n, DATA$. The functions $k_i, U$ can then be seen as projections to the $i$-th component.

The universe $\text{UNIV}(\Sigma_K)$ is an appropriate choice to be taken as standard semantics for a key signature $\Sigma_K$. Justification is given by the following properties of $\text{UNIV}(\Sigma_K)$:

- "possible objects" are identified by their data observable under (composite) keys.
- the semantics $DATA$ of the underlying data layer is preserved (up to isomorphism).

From a mathematical point of view, there is also a nice category-theoretic characterization of $\text{UNIV}(\Sigma_K)$. Let us consider the category $\Sigma_K$-alg[DATA] which is defined as follows:

1. The objects of $\Sigma_K$-alg[DATA] are $\Sigma_K$-algebras whose $\Sigma_D$-reducts are isomorphic to $DATA$.
2. The morphisms in $\Sigma_K$-alg[DATA] are those $\Sigma_K$-algebra morphisms $f : A \rightarrow B$ which are isomorphisms on the $\Sigma_D$-reducts of $A$ and $B$.

In connection with the assumptions on $DATA$ (see section 2), (1) and (2) guarantee that, for any objects $A, B \in \Sigma_K$-alg[DATA] there is exactly one isomorphism from the $\Sigma_D$-reduct of $A$ to the $\Sigma_D$-reduct of $B$. The uniqueness of isomorphisms on $\Sigma_D$-reducts is responsible for the fact that $\text{UNIV}(\Sigma_K)$ becomes a final object in $\Sigma_K$-alg[DATA].

**Theorem 4.1**

Let $\Sigma_K = \Sigma_D(S_0, R_K)$ be a key signature and $DATA$ be a fixed $\Sigma_D$-algebra satisfying the assumptions made in section 2. Then, $\text{UNIV}(\Sigma_K)$ is a final object in $\Sigma_K$-alg[DATA].

**Proof (sketch):**

Let $A$ be some algebra in $\Sigma_K$-alg[DATA]. By the assumption that identity is the only automorphism on $DATA$, there is a unique isomorphism $g$ from the $\Sigma_D$-reduct of $A$ to $DATA$. Now, we define a mapping $h : A \rightarrow \text{UNIV}(\Sigma_K)$ as follows. For each $s \in S$ and each $a \in A_s$,

$$h_s(a) = \lambda x[g(x_A(a))]$$

where $x$ ranges over $L_s$. Here, the interpretation $x_A$ of a key sequence $x \in L_s$ is given recursively by $x_A(a) = a$ and $(kx)_A(a) = y_A(k_A(a))$ for any key $k : s \rightarrow t$ and $y \in L_t$.

It is shown in Eh86 that $h$ is the only morphism from $A$ to $\text{UNIV}(\Sigma_K)$ in $\Sigma_K$-alg[DATA].
In some applications, the universe \( \text{UNIV}(\Sigma_K) \) is larger than desired. There may be objects in the universe which do not have real-world counterparts and which are, therefore, not really acceptable as "possible objects". With regard to the introductory example, employees, for instance, should earn more than $1000 per month. In our construction of the universe, however, there are also employees whose salary is smaller. Up to a certain point, such effects are unavoidable when modelling reality by formal means. We can get, however, much closer approximations to reality by allowing for constraints, i.e. conditions that constrain the set of possible objects in the universe.

Constraints can be expressed conveniently by first order formulas (with equality) over \( \Sigma_K \). As atomic formulas -first order formulas are built up from these in the ordinary way - we allow terms of sort \( \text{bool}^2 \) and expressions of the form \( l=r \) where \( l, r \) are terms of the same sort with function symbols from \( \Sigma_K \).

A \textit{key specification} is then a pair \( K=(\Sigma_K, C_K) \) where \( \Sigma_K \) is a key signature and \( C_K \) is a set of first order formulas over \( \Sigma_K \) called \textit{key constraints}.

The key constraints in \( C_K \) are meant to restrict the set of "possible objects" in our universe. On the other hand, there should not be excluded more objects than explicitly required. In more formal terms, the "largest" subalgebra of \( \text{UNIV}(\Sigma_K) \) satisfying the key constraints and preserving DATA would be really acceptable as standard semantics \( \text{UNIV}(K) \) of a key specification \( K \). Unfortunately, such a model does not always exist when allowing for arbitrary first order formulas to occur in \( C_K \). The construction of the universe works only for constraints that are in some sense compatible with morphisms in \( \Sigma_K\text{-alg}[\text{DATA}] \). The property which is needed is as follows.

Let \( \bar{x}=(x_1, \ldots, x_n) \) be an \( n \)-tuple of variables and let \( s_1 \ldots s_n \) be the string of their sorts. Let \( \varphi(\bar{x}) \) be a first order formula with free variables \( x_1, \ldots, x_n \). \( \varphi(\bar{x}) \) is called harmonic iff, for each morphism \( h:A \to B \) in \( \Sigma_K\text{-alg}[\text{DATA}] \) and all \( \bar{a} \in s_1, A \ldots x_n, A \) we have that \( A h \varphi(\bar{a}) \) implies \( B h \varphi(h(\bar{a})) \). A formula \( \psi \) is called universally harmonic iff it is of the form \( \psi=\forall \bar{x} \varphi(\bar{x}) \) having \( \varphi(\bar{x}) \) harmonic. If only one variable in \( \varphi(\bar{x}) \) is free (i.e. \( n=1 \)) \( \psi \) is called \textit{monadic}.

---

2) We assume the data layer to offer a predefined boolean data type.
Lemma 4.2
Let $A_i \subseteq \text{UNIV}(\Sigma_K)$, $i \in I$, such that the $\Sigma_0$-reducts of all $A_i$ are DATA (up to isomorphism). Then, the union $A = \bigcup_{i \in I} A_i$ is a subalgebra of $\text{UNIV}(\Sigma_K)$ whose $\Sigma_0$-reduct is DATA.

Lemma 4.3
Let $A_i$, $i \in I$, and $A$ as above. Let $\psi$ be a monadic universally harmonic formula. If $A_i = \psi$ for all $i \in I$, then $A = \psi$.

Proof: Let $\psi = \forall x \varphi(x)$, $x$ of sort $s$, $a \in s_A$. Then, there is an $i \in I$ such that $a \in s_{A_i}$. From $A_i = \psi$ we conclude $A_i = \psi(a)$. By $A_i \subseteq A$, the embedding is an injective morphism from $A_i$ to $A$. Since $\psi$ is harmonic, we have $A = \psi(a)$. This holds for all $a \in s_A$ so that $A = \psi$.

The above lemmas suggest to restrict to key constraints that are described by monadic universally harmonic formulas. When doing so, for any key specification $K = (\Sigma_K, C_K)$, the universe $\text{UNIV}(K)$ can be defined properly as the union $A$ of all subalgebras $A_i$ of $\text{UNIV}(\Sigma_K)$ which have $A_i = C_K$ and whose $\Sigma_0$-reducts are DATA (up to isomorphism)\(^3\). In this way, by Lemma 4.2 and 4.3, $\text{UNIV}(K)$ becomes the largest subalgebra of $\text{UNIV}(\Sigma_K)$ which satisfies $C_K$ and preserves DATA. Especially, it meets the requirement that the universe associated to a key specification should subsume all objects in $\text{UNIV}(\Sigma_K)$ which are not excluded explicitly by the key constraints.

Again, there is also a category-theoretic characterization of $\text{UNIV}(K)$ using finality. Let $K\text{-alg[DATA]}$ denote the full subcategory of $\Sigma_K\text{-alg[DATA]}$ of all $\Sigma_K$-algebras satisfying the formulas in $C_K$.

Theorem 4.4
Let $K = (\Sigma_K, C_K)$ be a key specification where $C_K$ is a set of monadic universally harmonic formulas $\forall x \varphi(x)$ having $x$ as object variable. Then, $\text{UNIV}(K)$ exists and is a final object in $K\text{-alg[DATA]}$.

For a proof, see Eh86.

\(^3\) Whenever, for all elements $\forall x \varphi(x)$ in $C_K$, $x$ is an object variable (in accordance with the intention that key constraints should restrict the set of possible objects), $\text{UNIV}(K)$ does exist. This is because the subalgebra of $\text{UNIV}(\Sigma_K)$ having empty carrier sets for all object sorts satisfies the key constraints.
As a serious drawback, it is often difficult to see whether or not a formula is harmonic. To overcome this problem (at least partially) we give a syntactical characterization of a class of formulas that is guaranteed to have the desired property.

A formula is called \textbf{positive} if it is built from atomic formulas by $\wedge$, $\vee$ and $\exists$, i.e. there is no negation $\neg$ and no $\forall$.

\textbf{Theorem 4.5:} Positive formulas are harmonic.

\textbf{Proof:} easy induction on the construction of positive formulas.

To be on the safe side, we suggest using key constraints of the form $\forall x \varphi(x)$ where $\varphi(x)$ is a monadic positive formula and $x$ an object variable. Then, the standard universe $\text{UNIV}(K)$ is well-defined.

\textbf{Example:} Referring to our employee/company example again, the following constraints (understood to be universally quantified) are positive and, thus, harmonic.

\begin{itemize}
  \item[a)] $\text{salary}(\text{manager}(c)) > 3000$
  \item[b)] $\text{sales}(c) > 1000 \times \text{salary}(\text{manager}(c))$
  \item[c)] $\text{manager}(\text{works-for}(e)) = e \land \exists e': \text{salary}(e') > \text{salary}(e)$
\end{itemize}

All formulas above assume the greater predicate $>$ to be offered by the data layer. a) and b) say that salaries of managers are bounded by a minimum amount of 3000 and by the sales of the company they work for. c) expresses that a manager makes more money than other employees in the same company. c) is equivalent to $\text{manager}(\text{works-for}(e)) \land e \implies \exists e': \text{salary}(e') > \text{salary}(e)$. ***

It should be mentioned that there are some aspects in modeling reality which cannot be taken into account so far. Key specifications (in the above sense) do not offer any structural mechanisms; in reality, however, one often has to do with set valued keys, generalization hierarchies and so on. It will be subject of future research to provide adequate facilities.

5. SCHEMA SPECIFICATIONS

The purpose of a conceptual schema is to characterize the structural and behavioural aspects of a database application. To complete the object layer, key specifications are extended to so-called schema specifications that are meant to describe the
conceptual schema of a database in a formal and implementation independent way. Roughly speaking, the design of a conceptual schema is separated into two extension steps:

\[ \text{D} = (\Sigma_D, E_D) \rightarrow \text{K} = (\Sigma_K, C_K) \rightarrow \text{SCH} = (\Sigma, C) \]

The design starts from a fixed data algebra \( \text{DATA} \) (up to isomorphism) specified by \( \text{D} \). The semantics of the first extension step, relative to the given data layer, is discussed in detail in the previous section. In the following, we give a brief account of the concepts involved in the second extension step completing the specification of the conceptual schema.

The schema signature \( \Sigma \) is to describe the global information structure of the database. Apart from the keys, it has to include additional attributes, object functions, and possibly predicates on objects. In formal terms,

\[ \Sigma = \Sigma_K + (\emptyset, R_R) \]

where \( R_R \) is a set of non-key operations together with their domains and codomains. Note that, at this specification level, object type constructors are permitted to obtain sets of objects, cartesian products of objects, etc.

Example: In the world of employees and companies, the following non-key information may be of interest

- superior: EMPLOYEE \( \rightarrow \) EMPLOYEE
- age: EMPLOYEE \( \rightarrow \) Nat
- birthdate: EMPLOYEE \( \rightarrow \) Date
- employs: COMPANY \( \rightarrow \) set EMPLOYEE

Based on the schema signature \( \Sigma \), a set \( C \) of static and dynamic integrity constraints can be expressed. This is done by means of temporal formulas over \( \Sigma \), i.e., first order formulas which may have temporal quantifiers \( \Box \) ("always") and \( \Diamond \) ("sometime"). Whenever there is no temporal quantification, a formula in \( C \) is understood to be a static constraint, otherwise it is a dynamic constraint.

Example: In a world of employees and companies, the following constraints may be given.

a) \( \forall c \forall e: \text{manager}(c) = e \implies \text{works-for}(e) = c \)

b) \( \forall e: \text{cname(works-for}(e)) = \text{IBM} \implies \text{salary}(e) > 2000 \)

c) \( \forall e: \text{age}(e) > 20 \)

d) \( \Box \forall e \forall c: \text{(works-for}(e) = c \implies \Diamond \lnot \text{works-for}(e) = c) \)

e) \( \Box \forall e \forall c: \text{(works-for}(e) = c \implies \Diamond \text{manager}(c) = e) \)
a) requires that managers are employed by the company they manage. b) says that employees of IBM have a salary of more than 2000 per month. By c), companies can only employ persons that are older than 20 years. d) requires that each employee will leave his company sometime. e) expresses that each employee becomes a manager in the course of time.

Note carefully that a), b) are first order formulas over the key signature. Nevertheless, they are not monadic universally harmonic formulas and can, therefore, not be taken as key constraints.

***

In the following, real world situations are modelled by Σ-algebras. To meet our intention, we have to restrict ourselves to those Σ-models which preserve DATA, which only have possible objects and which satisfy the static integrity constraints. The dynamic integrity constraints then describe admissible sequences of real world situations.

Given a schema specification SCH=(Σ, C) where C consists of a set C_s of static and a set C_d of dynamic constraints, i.e. C=C_s ∪ C_d. A Σ-situation d is a Σ-algebra such that d|Σ ≥ DATA, d|Σ ≥ SUNIV(K), and d|Σ ≥ C_s. d|Σ ≥ K is called the population of d.

A Σ-situation run is an infinite sequence d=(d_0, d_1, d_2, ...) of Σ-situations d_i (i ∈ N_0) such that d_i ∈ C_d.

The set of all Σ-situation runs is considered to be the semantics of SCH.

With regard to prototyping schema specifications, Σ-situations can be represented conveniently by so called Σ-states. A Σ-state is a set of closed atomic formulas asserting elementary facts like ename(e)='Smith', age(e)=30, works-for(e)=c, cname(c)='IBM', etc. where e, c are constants of type EMPLOYEE and COMPANY respectively. Admissibility of a given state sequence can be checked by methods for supervising and enforcing constraints as proposed in ELGB4, LEG85, LSE85. All these methods presume "total knowledge". Eh86 suggests to allow Σ-states given by arbitrary closed first order formulas. It is an open problem so far, how integrity checking can be performed when "partial knowledge" is involved.

6. CONCLUSIONS

This paper concentrates on a specific aspect of database semantics, namely the construction of a standard universe of "possible objects" and its abstract algebraic characterization (up to isomorphism) as a final algebra in an appropriate category. We feel
that such a construction is needed in order to provide a clean and precise semantic framework for characterizing admissible states and state sequences. The latter issue is sketched only briefly in this paper and clearly needs more elaboration.

Our approach is neither model theoretic nor proof theoretic, but shows aspects of both lines of thought. As in the proof theoretic approach, we define a database state to be a set of sentences (formulas), thus representing a state of knowledge rather than a situation of the world to be represented. Situations, in turn, are models, and we take care to give a model theoretic characterization of admissible states and state sequences. Identifying situations and database states works only in special cases where it is assumed that the state uniquely determines a situation. In order to achieve this, additional assumptions like negation as failure [C178], closed world assumption [Re78], or circumscription [Mc80] have to be made. These approaches, however, cannot handle partial and disjunctive information appropriately. Nevertheless, they constitute a very important standard case, and it would be interesting to give a model theoretic characterization of these assumptions in our framework. This is subject to further study.

REFERENCES


DMW82 Dosch, W./Mascari, G./Wirsing, M.: On the Algebraic Specification of Databases. Proc. 8th Int. Conf. on Very Large Data Bases, Mexico City 1982


Eh86 Ehrich, H.-D.: Key Extension of Abstract Data Types, Final Algebras, and Database Semantics. Bericht Nr. 86-01, TU Braunschweig, 1986


Ja85 Jantke, K.: The Recursive Power of Algebraic Semantics (submitted for publication)


LSE85 Lipeck, U.W./Saake, G./Ehrich, H.-D.: Monitoring Dynamic Integrity Constraints by Transition Graphs (submitted for publication)


