Abstract

The algebraic specification of abstract data types provides a number of features for structuring specifications in order to make them easier to write, to read, to understand, and to maintain. Among the most important such features are different kinds of extensions, based on initial, final, or behavioural semantics. This paper studies a new kind of extension called key extension, and its final algebra semantics. Key extensions model one of the essential steps in database specification where abstract object types are to be specified on the basis of given abstract data types. The intended standard semantics is a universe of "possible objects" that provides the basis for further database concepts like situations, states, etc. It is shown under which conditions final algebras exist that can serve as a natural standard semantics for key extensions. We also characterize rather a large class of constraints that can be used for keys in accordance with our semantics.
1. Introduction

While the field of programming language semantics is by now well matured, involving a high degree of formal concepts and sophisticated mathematical structures, the field of database semantics still appears to be in a rather informal state. The purpose of this paper is to apply algebraic notions to the database field that have proven fruitful in the theory of abstract data types.

To this end, an approach to database specification is taken that employs notions like signature and algebra as known from abstract data type theory. Our approach is different from those in DMW82 and EKW78 where applicative concepts are uniformly applied. We favor a modal approach as outlined in GMS83 where database states are algebras rather than elements in a carrier of an algebra.

A particular aspect of our approach is the separation of data and objects on different levels of description. Data are storeable and printable items used to describe properties of objects while objects are entities to be described. Eh84 gives a short account of the basic ideas. For the moment being, we have ignored the problem of specifying actions on a database. KMS85 addresses this problem and develops ideas that can be used to supplement our approach.

The basic separation of data and objects is refined further in this paper by separating the object specification into two steps, the first of which is the specification of a key system for the objects. The present paper concentrates on this first step, considered as an extension of the data level, and suggests an algebraic semantics for it in terms of final algebras. The semantics associates a universe of "possible objects" with the object sorts.
The specification of keys may incorporate certain constraints. We characterize rather a large class of constraints that can be used for keys in accordance with our semantics.

In order to put our results into perspective, we briefly summarize basic notions of the algebraic theory of abstract data types in the next section, emphasizing the importance of data type extensions. In section 3, the basic construction of the universe is given, and its finality in an appropriate category is proved. In section 4, we modify this construction in order to incorporate constraints on keys. A large class of constraints called positive formulas is characterized syntactically in section 5, and it is shown that positive constraints are guaranteed to make our construction work. Finally, in section 6, we give a brief survey of the remaining step to complete a database schema specification.

2. Extensions of abstract data types

The algebraic theory of abstract data types has been studied intensively since the pioneering paper GTW78. An introduction is given in the recent textbook EM85. Extensions of abstract data types play a dominant role (Eh78, Wa79) because they constitute one of the main structuring principles for algebraic specifications. There are several meanings that can be associated with an extension: initial or free semantics (GTW78, Eh82), final semantics (Wa79, Ja85, Bo85) and behavioural semantics (HR83) are the most well known approaches.

We briefly review a few fundamental concepts and facts about abstract data types and extensions.

A signature $\Sigma = (S, R)$ consists of a set $S$ of sorts and an $S^* \times S^*$-indexed set family $R$ of operators. If $\omega \in R_{x,s}$ and $x = s_1 \ldots s_n$, we write $\omega : s_1 * \ldots * s_n \rightarrow s$. 
A \( \Sigma \)-algebra \( A \) consists of a set \( s_A \), the carrier of sort \( s \), for each \( s \in S \), and an operation \( \omega_A : s_{1A} \times \cdots \times s_{nA} \to s_A \) for each operator \( \omega : s_1 \times \cdots \times s_n \to s \) in \( \Omega \).

A \( \Sigma \)-algebra-morphism \( h : A \to B \) is an \( S \)-indexed family of mappings \( h_s : s_A \to s_B \) such that, for each \( \omega : s_1 \times \cdots \times s_n \to s \), \( a_1 \in s_{1A}, \ldots, a_n \in s_{nA} \), the morphism condition \( h_s(\omega_A(a_1, \ldots, a_n)) = \omega_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \) holds. The category of all \( \Sigma \)-algebras and all \( \Sigma \)-algebra morphisms is denoted by \( \Sigma \text{-alg} \).

It is well known that \( \Sigma \text{-alg} \) has an initial object, i.e. an algebra from which there is exactly one morphism to any algebra in \( \Sigma \text{-alg} \). \( \Sigma \text{-alg} \) also has a final object, i.e. an algebra to which there is exactly one morphism from any algebra in \( \Sigma \text{-alg} \), but this final algebra is degenerate and of no particular interest.

A specification \( D = (\Sigma, E) \) consists of a signature \( \Sigma \) and a set \( E \) of \( \Sigma \)-formulas as axioms. For the sake of simplicity, we only consider \( \Sigma \)-equations as axioms, in accordance with most of the literature. A specification \( D \) determines the full subcategory \( D \text{-alg} \subseteq \Sigma \text{-alg} \) of all algebras satisfying all equations in \( E \).

\( D \text{-alg} \) also has initial and final objects. The initial algebra is taken as standard semantics for \( D \) in the initial-algebra approach. The final algebra in \( D \text{-alg} \) coincides with that in \( \Sigma \text{-alg} \).

A signature morphism \( f : \Sigma_1 \to \Sigma_2 \) is a mapping from sorts to sorts and operators to operators such that, if \( \omega : s_1 \times \cdots \times s_n \to s \) in \( \Sigma_1 \), then its image works on the images of the respective sorts, i.e. \( f(\omega) : f(s_1) \times \cdots \times f(s_n) \to f(s) \). A signature morphism \( f : \Sigma_1 \to \Sigma_2 \) determines a forgetful functor \( \Sigma_2 \text{-alg} \to \Sigma_1 \text{-alg} \) by sending a \( \Sigma_2 \)-algebra \( A \) to \( \Sigma_1 \text{-alg} \) by sending a \( \Sigma_2 \)-algebra \( A \) to \( \Sigma_1 \text{-alg} \) by sending a \( \Sigma_1 \)-algebra \( A \) to \( f(A) \) where \( s_f(A) = f(s)_A \) and \( \omega_f(A) = f(\omega)_A \). \( \Sigma_2 \)-algebra morphisms \( h : A \to B \) are sent to \( \Sigma_1 \)-algebra morphisms \( f(h) : f(A) \to f(B) \) where \( f(h)_s = h_f(s) \). A specification morphism \( f : D_1 \to D_2 \), \( D_i = (\Sigma_i, E_i) \) for \( i = 1, 2 \), is a signature morphism \( f : \Sigma_1 \to \Sigma_2 \) such that \( f(D_2 \text{-alg}) \subseteq D_1 \text{-alg} \).
An extension is, most generally, some translation from $\Sigma_1$-algebras to $\Sigma_2$-algebras, reversing the direction of $\mathcal{F}$. On the level of abstract data types, i.e. classes of algebras, $\mathcal{F}^{-1}$ provides such a translation. A well-known functorial extension is given by the left adjoint $f^*:\Sigma_1\text{-alg} \rightarrow \Sigma_2\text{-alg}$ of $\mathcal{F}$, sending each $\Sigma_1$-algebra to the free $\Sigma_2$-algebra over it wrt $f$.

The standard case of an extension is specified by an inclusion $f:D_1 \hookrightarrow D_2$ that is a specification morphism. This models the process of adding new types to an existing type structure. In the initial-algebra approach, the semantics of $f$ is $f^*$. This is in accordance with the fact that left adjoints preserve initiality. The idea that a data type $A$ specified by $D_1$ should "persist" in its extension leads to the requirement that $\mathcal{F}(f^*(A))$, roughly speaking, should be isomorphic to $A$ in a natural way.

Wand has shown (Wa79) that, among the "legal implementations" of an extension $f$ (which is a category of $D_2$-algebras sent to initial $D_1$-algebras by $\mathcal{F}$) has a final algebra. He suggests to take this as the standard semantics of the extension $f$, thus establishing the basis for final algebra semantics.

The "legal implementations" defined by Wand have in a sense equivalent behaviour. HR83 discusses several notions of behavioural equivalence of data types suggesting that the semantics of an extension $f$ should be such a (polymorphic) equivalence class.

All notions of extension studied in the framework of abstract data types are constructive in the sense that the extended type is in some way generated by the type specified by $D_1$. Technically, they are based on the free extension $f^*$. In this respect, key extensions to be introduced in the next section are different.

For what follows, we assume a monomorphic abstract data type $\text{DATA}$ to be given. Let $\Sigma_D = (\mathcal{S}_D, \mathcal{R}_D)$ be its signature. Our results are based on the following assumptions about $\text{DATA}$:
1. all carriers $s_{\text{DATA}}, s \in S_D$, 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 and can, for all intents and purposes, simply be omitted. Moreover, most practical data types are minimal algebras (for instance, initial algebras are minimal), and these never have proper automorphisms.

3. Key extensions

In a database environment, data play the role of descriptors for properties of objects, e.g. name, birthdate, salary of a person or colour, weight, price of a part. Technically speaking, data are values of attributes, and attributes are mappings from object types to data types. Moreover, attributes are the only means for accessing data from objects.

In the relational model of data, attributes are the only functions applicable to objects. Consequently, objects are uniformly represented by their "tuples", i.e. the lists of their attribute values. Other data models provide the concept of "relationship", e.g. father, spouse or sibling as relationships between persons.

Given a data signature $\Sigma_D = (S_D, R_D)$ with a fixed interpretation DATA (up to isomorphisms), the specification of object types in a conceptual database schema consists of

- a set $S_O$ of object sorts
- an $S^*S$-indexed set family $R_O$ of object functions
where $S = S_D \cup S_0$. Object functions with a data sort as target sort represent attributes, the others functional relationships.

Specifying the object level on top of a given data level can be a very complex task, so that it is best separated into a sequence of smaller steps. Here we concentrate on the first step which we call key extension. Subsequent steps are surveyed in section 5.

This first step consists of identifying the objects that can possibly be dealt with in a database application. This is done by determining keys, i.e. object functions that are meant to identify an object uniquely.

**Definition 3.1:** Let $s \in S_0$. A key (of $s$) is a unary object function $k : s \rightarrow t$ where $t \in S$.

Examples of keys are social security numbers of persons, part numbers of parts, names of projects, etc. In many examples, there is only one key for each object sort, and this key is an attribute. This is, however, not necessarily so. There can be several keys that together identify an object, e.g. name, birthdate and address of a person, and there can be object-valued keys like father, affiliation, etc. Sometimes data and object keys are mixed, for example if a person is identified by his or her name and father.

The first step in specifying the object level consists of defining the keys, that is to provide

- the complete set $S_0$ of object sorts
- an $S_0 \times S$-indexed set family $R_K$ of keys.

The resulting signature $\mathcal{I}_K = \mathcal{I}_D + (S_0, R_K)$ is called the key signature.
Key signatures can be conveniently visualized by their signature graphs. We represent data sorts by squares, object sorts by circles, and keys \( k : s \rightarrow t \) by arcs from \( s \) to \( t \).

**Example 3.2:**

The intended interpretation of a key signature is a universe of "possible objects" that can be identified in the specified key system. Technically speaking, this is a \( \Sigma_k \)-algebra having the property that its \( \Sigma_D \)-reduct is (isomorphic to) the given data algebra \( \text{DATA} \). But there are, in general, many different such algebras. How can we agree on a "canonical" one? Studying simple examples will give us the right idea.

If we have just one data key \( k : s \rightarrow r \), \( r \in S_D \), then the candidates for a universe of "possible objects" are \( \Sigma_k \)-algebras \( A \) consisting of \( \text{DATA} \) enriched by a carrier set \( s_A \) for \( s \) and a mapping \( k_A : s_A \rightarrow r_A \) where \( r_A = r_{\text{DATA}} \). The idea of a key identifying possible objects suggests that \( k_A \) be bijective. Especially, identity \( k_A = \text{id} \), thus \( s_A = r_A \), would be a suitable choice.

More generally, if we have \( n > 0 \) data keys \( k_i : s \rightarrow r_i \), \( r_i \in S_D \), \( 1 \leq i < n \), the appropriate choice would be \( s_A = r_1^A \times \cdots \times r_n^A \) where \( k_i \) is the \( i \)-th projection. This corresponds to general assumptions usually made in the relational data model where the cartesian product of all (key) attribute value domains is the set of all possible tuples (i.e. objects).
The matter becomes more difficult, however, if we allow for object keys. In order to generalize the above considerations, we look for a universal algebraic property characterizing our choices.

Let us consider the category \( \Sigma_K\text{-alg}[\text{DATA}] \) defined as follows. The objects of this category are \( \Sigma_K\)-algebras whose \( \Sigma_D\)-reducts are isomorphic to \( \text{DATA} \), and the morphisms are those \( \Sigma_K\)-algebra morphisms whose \( \Sigma_D\)-reducts are isomorphisms on \( \text{DATA} \).

If we have only data keys, our choice made above, i.e. the cartesian product, is easily seen to be a final algebra in this category: any object in any other algebra must be mapped by any morphism to that object with the same (or rather isomorphic) attribute values for all attributes. There is exactly one morphism doing this.

Thus, if all keys are attributes, \( \Sigma_K\text{-alg}[\text{DATA}] \) has a final object, and this is our favorite choice.

It is the first main result presented in this paper that this observation generalizes to arbitrary key signatures.

**Theorem 3.3:** Let \( \Sigma_K = \Sigma_D + (S_0, \Pi_K) \) be a key signature, and let \( \text{DATA} \) be a fixed \( \Sigma_D\)-algebra satisfying the assumptions made in section 2. Then \( \Sigma_K\text{-alg}[\text{DATA}] \) has a final algebra.

The proof is constructive: we construct a specific \( \Sigma_K\)-algebra \( \text{UNIV}(\Sigma_K) \) and show that it is final. The idea is that \( \text{UNIV}(\Sigma_K) \) consists of all object descriptions that are possible in the given key signature \( \Sigma_K \). The construction of \( \text{UNIV}(\Sigma_K) \) requires some preparation.

**Definition 3.4:** 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 $k:s \rightarrow u$ 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 $t$ are directed paths from node $s$ to node $t$. Let $L_{s,t}$ denote the set of all key sequences from $s$ to $t$.

An object of sort $s$ can be characterized by all its "observations", i.e. all values obtained under key sequences leading to a data sort. Accordingly, we define

$$L_s := \bigcup_{r \in S_D} L_{s,r}$$

Example 3.5: Referring to example 3.2, we have

$$L_{s1} = (k_1k_2k_3)^*k_4 \cup k_1[(k_2k_3k_1)^*k_5 \cup k_2(k_3k_1k_2)^*k_6]$$

Remark 3.6: For $r \in S_D$, we have $L_{r,r} = \{\varepsilon\}$ and $L_{s,r} = L_{r,s} = \emptyset$ for $s \neq r$. Thus, $L_r = \{\varepsilon\}$.

Given a key signature $\Sigma_K$, our universe $\text{UNIV}(\Sigma_K)$ is constructed as follows. For brevity, we write $U$ for $\text{UNIV}(\Sigma_K)$. The carrier set of sort $s \in S$ is

$$s_U = \{\psi \mid L_s \longrightarrow \bigcup_{r \in S_D} r_{\text{DATA}}, \psi(L_s,r) \in r_{\text{DATA}} \text{ for each } r \in S_D\}$$

Provided that $r_{\text{DATA}} \neq \emptyset$ for each $r \in S_D$ (which we assume), we have $s_U \neq \emptyset$ for each $s \in S$, for if $L_s = \emptyset$, then $s_U$ contains just one mapping, the empty mapping.
For each data sort \( r \in S_D \), we have \( L_r = \{ \varepsilon \} \), i.e. \( s_U = \{ \{ (\varepsilon, a) \} \mid a \in r_{\text{DATA}} \} \). There is an obvious 1-1-correspondence \( h: r_{\text{DATA}} \rightarrow r_U \), \( h(a) = (\varepsilon, a) \) (actually \( \{ (\varepsilon, a) \} \)), but we omit the set braces.

The operations of \( U \) are defined as follows. For data operators \( \omega: s_1 \ast \ldots \ast s_n \rightarrow s_0 \), we define

\[
\omega_U((\varepsilon, a_1), \ldots, (\varepsilon, a_n)) = (\varepsilon, \omega_{\text{DATA}}(a_1, \ldots, a_n)).
\]

From this, \( h \) is easily seen to give an isomorphism from \( \text{DATA} \) to the \( \Sigma_D \)-reduct of \( U \).

For keys \( k: s \rightarrow t \) we define

\[
k_U(\varphi) := \lambda x \ [\varphi(kx)]
\]

where \( x \) ranges over \( L_t \). This means that each mapping \( \varphi \in s_U \) is sent to that mapping \( \varphi' \in t_U \) satisfying \( \varphi'(x) = \varphi(kx) \) for each \( x \in L_t \). If \( t \in S_D \), we get \( k_U(\varphi) = (\varepsilon, \varphi(k)) \) where \( \varphi(k) \in t_{\text{DATA}} \), since \( L_t = \{ \varepsilon \} \).

The following lemma proves theorem 3.3.

**Lemma 3.7**: \( \text{UNIV}(\Sigma_K) \) is a final object in \( \Sigma_K\text{-alg}[\text{DATA}] \).

**Proof**: Let \( A \) be some algebra in \( \Sigma_K\text{-alg}[\text{DATA}] \). We define a mapping \( h: A \rightarrow U \) as follows, where again \( U = \text{UNIV}(\Sigma_K) \). For each \( s \in S \) and each \( a \in s_A \), let

\[
h_s(a) = \lambda x \ [x_A(a)],
\]

where \( x \) ranges over \( L_s \). Here, the interpretation \( x_A \) of a key sequence \( x \in L_s \) is given by \( \varepsilon_A(a) = a \) and \( (kY)_A(a) = y_A(k_A(a)) \).

Thus, for \( r \in S_D \), \( h_r(a) = (\varepsilon, a) \). This shows that \( h \) is an isomorphism on \( \text{DATA} \).
In order to show that $h$ is a morphism, we have to show compatibility with keys. Let $k:s \rightarrow t$ be a key, $a \in s_A$, and $x \in L_t$. Then we have

$$k_U(h_s(a))(x) = h_s(a)(kx) = (kx)_A(a) = x_A(k_A(a)) = h_t(k_A(a))(x)$$

Consequently, $k_U(h_s(a)) = h_t(k_A(a))$ proving that $h$ is a morphism.

In order to show finality of $U$, we have to show that $h$ is the only morphism from $A$ to $U$.

Let $h':A \rightarrow U$ be an arbitrary morphism. By definition, $h'$ and $h$ are isomorphisms on the DATA part of $A$. Since DATA has no automorphisms, we have for each $r \in S_D$ and each $a \in r_A$:

$$h'_r(a) = h_r(a) = (\varepsilon, a).$$

For $s \in S_0$, we prove $h'_s(a) = h_s(a)$ by induction on the length of key sequences. Let $k:s \rightarrow t$ be a key, $y \in L_t$ a key sequence, and $x = ky$. Assume that $h'_s(b)(y) = h_s(b)(y)$ for all $b \in s_A$ (which holds for $y = \varepsilon$ as shown above). Then we conclude for $a \in s_A$:

$$h'_s(a)(x) = k_U(h'_s(a))(y) = h'_s(k_A(a))(y) = h_s(k_A(a))(y) = k_U(h_s(a))(y) = h_s(a)(x)$$

Since this holds for all $s \in S_0$, all $a \in s_A$ and all $x \in L_S$, we have $h'_s = h_s$, concluding the proof.

Actually, we have proved the more general result that for each automorphism on DATA there is exactly one morphism $h:A \rightarrow DATA$ sending data values to their images under the automorphism.
4. Key constraints

In some applications, the universe $\text{UNIV}(\Sigma_K)$ constructed in the previous section is larger than desired. There may be objects in the universe that do not have real-world counterparts so that they are not really acceptable as "possible" objects.

For example, if a person is identified by his or her name, age and father, the age should be somewhat less than the father's age. In our construction of the universe, however, there would be persons older than their father.

Up to a certain point, such effects are unavoidable when modelling reality by formal means. We can obtain, however, much closer approximations to reality by allowing for constraints, i.e. conditions that constrain the set of possible objects in the universe.

In this section, we give a class of constraints that permit a standard construction of a suitable universe satisfying the constraints. The construction again utilizes finality.

Constraints are described by first-order formulas (with equality) over $\Sigma_K$. The construction of the universe works only for constraints that are in a sense compatible with morphisms in $\Sigma_K\text{-alg}[\text{DATA}]$. The property required is as follows. Let $\vec{x} = (x_1, \ldots, x_n)$ be an $n$-tuple of variables, and let $y = s_1 \ldots s_n$ be the string of their sorts. Let $\psi(\vec{x})$ be a first-order formula with free variables $x_1, \ldots, x_n$.

**Definition 4.1**: A formula $\psi(\vec{x})$ is called harmonic iff, for each morphism $h: A \rightarrow B$ in $\Sigma_K\text{-alg}[\text{DATA}]$ and all $\vec{a} \in \gamma_A$, we have $A \models \psi(\vec{a})$ implies $B \models \psi(h(\vec{a}))$. A formula $\gamma$ is called universally harmonic iff it is of the form $\gamma = \forall \vec{x} \; \psi(\vec{x})$ and $\psi(\vec{x})$ is harmonic. A set $C$ of formulas is called universally harmonic iff all formulas in $C$ have this property.
If \( n = 1 \), i.e. there is just one variable, the formula or set of formulas, respectively, is called monadic.

**Definition 4.2:** A key specification is a pair \( K = (\Sigma_K, C_K) \) where \( \Sigma_K \) is a key signature and \( C_K \) is a set of first-order formulas called key constraints.

**Definition 4.3:** Let \( K = (\Sigma_K, C_K) \) be a key specification. By \( K\text{-alg}[\text{DATA}] \) we denote the full subcategory of \( \Sigma_K\text{-alg}[\text{DATA}] \) of all \( \Sigma_K \)-algebras satisfying all formulas in \( C_K \).

The main result of this section is the following.

**Theorem 4.4:** Let \( K \) be a key specification, and let \( \text{DATA} \) satisfy the assumptions made in section 2. If \( C_K \) is a monadic universally harmonic set of formulas, then \( K\text{-alg}[\text{DATA}] \) has a final algebra.

The proof of the theorem is again constructive: we construct a specific \( K \)-algebra \( \text{UNIV}(K) \) and show that it is final. The first step in constructing \( \text{UNIV}(K) \) is to construct \( \text{UNIV}(\Sigma_K) \) as described in the last section. Then, \( \text{UNIV}(K) \) is defined to be a specific subalgebra of \( \text{UNIV}(\Sigma_K) \), namely the largest subalgebra that satisfies the constraints. The next lemmas show that it exists.

**Lemma 4.5:** Let \( A_i \in \text{UNIV}(\Sigma_K) \), \( i \in I \), such that the \( \Sigma_D \)-reducts of all \( A_i \) are \( \text{DATA} \). Then the union \( A = \bigcup_{i \in I} A_i \) is a subalgebra of \( \text{UNIV}(\Sigma_K) \).

**Proof:** That \( A \) is closed with respect to all operations is obvious from the fact that we only have unary operations in the object parts where the \( A_i \) differ.

**Lemma 4.6:** Let \( A_i \), \( i \in I \), and \( A \) be as in lemma 4.5. Let \( \psi \) be a monadic universally harmonic formula. Then, if \( A_i \models \psi \) holds for all \( i \in I \), then \( A \models \psi \).
Proof: Let $\psi = \forall x \varphi(x)$, x of sort s, $a \in \mathbb{S}_A$. Then there is an $i \in I$ such that $a \in \mathbb{S}_{A_i}$. From $A_i \models \psi$ we conclude $A_i \models \varphi(a)$. Since $\varphi$ is harmonic and $A_i \subseteq A$, we have $A \models \varphi(a)$. Since this holds for all $a \in \mathbb{S}_A$, we conclude $A \models \psi$.

**Definition 4.7:** Let $K$ be a key specification with a monadic universally harmonic set $C_K$ of key constraints. Then, $\text{UNIV}(K)$ is defined to be the union of all subalgebras of $\text{UNIV}(\Sigma_K)$ satisfying $C_K$.

That means that $\text{UNIV}(K)$ is the largest subalgebra satisfying $C_K$. If $C_K$ contains non-monadic formulas, $\text{UNIV}(K)$ can be constructed this way, too, but it need not satisfy $C_K$. As a counterexample, consider $\forall x \forall y |\text{birthdate}(x) - \text{birthdate}(y)| < 10 \text{ years}$ which may very well hold in two subalgebras, but not in their union.

The following lemma proves the main result of this section.

**Lemma 4.8:** Let $K$ be as in definition 4.7. Then $\text{UNIV}(K)$ is final in $K\text{-alg}[\text{DATA}]$.

Proof: Let $\gamma \in C_K$, $\gamma = \forall x \varphi(x)$, x of sort s. Let A be any algebra in $K\text{-alg}[\text{DATA}]$, i.e. $A \models \gamma$. Let $h : A \rightarrow \text{UNIV}(\Sigma_K)$ be the final morphism in $\Sigma_K\text{-alg}[\text{DATA}]$. Let $a \in \mathbb{S}_A$. Since $A \models \varphi(a)$ and $\varphi$ is harmonic, we have $\text{UNIV}(\Sigma_K) \models \varphi(h(a))$. Since this holds for all $a \in \mathbb{S}_A$, we conclude $h(A) \models \gamma$ where $h(A) \subseteq \text{UNIV}(\Sigma_K)$ is the morphic image of A. Since this holds for all $\gamma \in C_K$, we have $h(A) \subseteq K\text{-alg}[\text{DATA}]$ and consequently $h(A) \subseteq \text{UNIV}(K)$. Thus, $h$ may be considered as a morphism $h : A \rightarrow \text{UNIV}(K)$. By uniqueness of $h$ in $\Sigma_K\text{-alg}[\text{DATA}]$, $h$ is unique in $K\text{-alg}[\text{DATA}]$. Thus, $\text{UNIV}(K)$ is final in $K\text{-alg}[\text{DATA}]$. 
5. Positive constraints

Our result shows that we have a final semantics for key specifications as long as we restrict ourselves to monadic universally harmonic constraints. We now characterize a class of formulas syntactically that is guaranteed to have the desired property.

Definition 5.1: A formula is called **positive** iff it is built from atomic formulas by \( \land, \lor \) and \( \exists \), i.e. there is no negation \( \neg \) and no \( \forall \).

Theorem 5.2: Positive formulas are harmonic.

Proof: Let \( h: A \to B \in \Sigma_K\text{-alg}[DATA] \). We prove the theorem by induction on the construction of positive formulas.

Let \( \bar{x} = (x_1, \ldots, x_n) \) be an \( n \)-tuple of variables of sorts \( y = s_1 \ldots s_n \).

1. Atomic formulas \( \psi(\bar{x}) \) are comparison expressions between terms. If \( A \models \psi(\bar{a}) \) for some \( \bar{a} \in \gamma_A \), then \( B \models \psi(h(\bar{a})) \) follows from the morphism property of \( h \).

2. Let \( \psi(\bar{x}) = \psi_1(\bar{x}) \land \psi_2(\bar{x}) \). If \( A \models \psi(\bar{a}) \), then both \( A \models \psi_1(\bar{a}) \) and \( A \models \psi_2(\bar{a}) \) hold. By assumption, \( \psi_1 \) and \( \psi_2 \) are harmonic, thus \( B \models \psi_1(h(\bar{a})) \) and \( B \models \psi_2(h(\bar{a})) \), hence \( B \models \psi_1(h(\bar{a})) \land \psi_2(h(\bar{a})) = \psi(h(\bar{a})) \). Consequently, \( \psi(\bar{x}) \) is harmonic.

3. The proof for \( \psi(\bar{x}) = \psi_1(\bar{x}) \lor \psi_2(\bar{x}) \) is analogous.

4. Let \( \psi(\bar{x}) = \exists v \psi_1(\bar{x}, v) \), \( v \) of sort \( t \). Let \( A \models \psi(\bar{a}) \) for some \( \bar{a} \in \gamma_A \). Then, \( A \models \psi_1(\bar{a}, b) \) for some \( b \in t_A \). By assumption, \( \psi_1 \) is harmonic, thus \( B \models \psi_1(h(\bar{a}), h(b)) \). It follows that \( B \models \exists v (\psi_1(h(\bar{a}), v)) \), hence \( B \models \psi(h(\bar{a})) \). Consequently, \( \psi(\bar{x}) \) is harmonic.
As a consequence of this result, we suggest using key constraints of the form $\forall x \psi(x)$ where $\psi(x)$ is a monadic positive formula. Then, the standard universe $\text{UNIV}(K)$ exists as constructed in the previous section.

Example 5.3: Let objects of type PERSON have keys name, age, father and spouse. The following constraints (understood to be universally quantified) are positive and thus harmonic.

1. $\text{age}(\text{father}(x)) > \text{age}(x)$
2. $\text{spouse}(\text{spouse}(x)) = x$
3. $\exists y \text{ age}(y) > \text{age}(x)$

The following constraints are not harmonic (so they have to be non-positive).

4. $x \neq \text{spouse}(x)$
5. $\text{spouse}(x) \neq \text{spouse}(\text{father}(x))$
6. $x \neq y \implies \text{spouse}(x) \neq \text{spouse}(y)$

The constraint

7. $\text{name}(x) \neq \text{name}(\text{spouse}(x))$, 

however, is harmonic. Whether it is positive or not depends on whether inequality is defined on the data type of names as a basic operation or not.

The last example shows that there are harmonic formulas which are not positive. Thus, theorem 5.2 may not be reversed. However, negation of atomic formulas with a data sort does no harm to harmonicity in general, since morphisms are isomorphisms on the data part which preserve not only equality but also inequality. We conjecture that these are the only cases where non-positive formulas are harmonic.
6. Database semantics

The purpose of a conceptual database schema specification is to characterize the admissible structures and behaviour of a database. For the moment being, we exclude the specification of actions, e.g. updates, from our consideration. We include, however, action-independent characterization of dynamic behaviour by specifying admissible state sequences. [Li85] shows how to transform such action-independent constraints to pre- and postconditions of actions in subsequent refinement steps of a design.

We propose to separate conceptual schema specification into two main extension steps:

\[
D = (\Sigma_D, E) \implies K = (\Sigma_K, C_K) \implies SCH = (\Sigma, C)
\]

The design is based on a fixed data algebra DATA (up to isomorphism) specified by \(D\). The semantics of the first extension step called key extension is discussed in the previous sections. Here we give a brief account of the concepts involved in the second extension step completing the specification of the conceptual schema.

The semantics of \(SCH\) relative to that of \(K\) is intended to be a class of admissible state sequences. States are "states of knowledge", i.e. more or less complete samples of assertions about "real world" situations.

Let \(K = (\Sigma_K, C_K)\) be a key specification, and let \(\text{UNIV}(K) \subseteq K\text{-alg}[\text{DATA}]\) be its semantics, i.e. a given standard universe. The schema signature

\[
\Sigma = \Sigma_K + (\emptyset, \Pi_R)
\]
provides additional attributes, object functions, and possibly predicates on objects and data.

Definition 6.1: A Σ-situation σ is a Σ-model such that σ|Σ₀ = DATA and σ|Σ肯 ∈ UNIV( ken ). σ|Σ肯 is called the population of σ.

Typically, states are sets of closed atomic formulas asserting elementary facts like name(p) = 'Meyer', age(p) = 32, father(p) = p', etc. for PERSON constants p, p'. In order to take more general forms of knowledge into account, e.g. rules, we admit arbitrary closed first-order formulas.

Definition 6.2: A Σ-state ST is a specification ST = (Σ, Z) where Z is a set of closed first-order formulas over Σ.

A Σ-state determines the class \([ST] = \{ \sigma | \sigma \models Z \}\) of situations. In the "classical" case assuming total knowledge, there is a unique (up to isomorphism) "smallest" situation σ_min ∈ [ST] (initial with respect to inclusion) which can be used to associate a "standard situation" with a state that is described by that state. In this case, states and situations can be identified. It is an interesting problem to characterize those states that allow for such a standard interpretation, e.g. by means of initiality.

In general, however, if the state contains partial knowledge, e.g. disjunctive or negative assertions, it is more appropriate to think of a state as a description of many possible situations.

It is obvious how "static" constraints in a schema specification, i.e. first-order formulas in C, constrain the class of admissible situations. For defining a state ST to be admissible, there are two possibilities, a "weak" one requiring that there is an admissible situation in [ST], and a "strong" one requiring that all situations in [ST] are admissible.
We want, however, to include also "dynamic" constraints characterizing admissible dynamic behaviour in terms of situation and state sequences.

Definition 6.3: A \( \Sigma \)-situation run is an infinite sequence \( \Sigma = (s_0, s_1, \ldots) \) of situations. A \( \Sigma \)-state run is an infinite sequence \( ST = (ST_0, ST_1, \ldots) \) of states.

A \( \Sigma \)-state run determines a class of \( \Sigma \)-situation runs by \( [ST] = \{ s \mid s \in [ST_i] \text{ for each } i \leq 0 \} \). For characterizing admissible \( \Sigma \)-situation runs, we use a temporal logic involving temporal quantifiers like always and sometime. Models of such formulas are \( \Sigma \)-situation runs so that we have

\[
[\text{SCH}] = \{ s \mid s \models C \}
\]

This, again, offers two possibilities for defining admissible state runs: we may call \( ST \) weakly admissible iff \( [ST] \) contains an admissible \( s \), i.e. \( [ST] \cap [\text{SCH}] \neq \emptyset \), and we may call \( ST \) strongly admissible iff all \( s \in [ST] \) are admissible, i.e. \( [ST] \subseteq [\text{SCH}] \). It requires further study to decide which notion is more appropriate for which purpose. In the "classical" case where states denote unique situations, both notions coincide.

One important aspect is to investigate which notion of admissibility can be effectively supported by methods supervising and enforcing constraints during database operation. For the "classical" case, the problem has been studied in [ELG84, LEB85, Li85, LSEB85, Sa85].
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


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)
