A concise formal framework for data modeling

A Concise Formal Framework for Data Modeling

by

A.T.M.Aerts and K.M. van Hee

89/12

December, 1989

COMPUTING SCIENCE NOTES

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
Editors: prof.dr.M.Rem
prof.dr.K.M. van Hee

A Concise Formal Framework for Data Modeling

A.T.M. Aerts and K.M. van Hee

Department of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven, The Netherlands

ABSTRACT

A framework for datamodeling is presented which is both formal and concise. It is constructed around the concepts of objects and their mutual functional relations, has a complete datalanguage based on first order logic and a diagram technique. Relations to other frameworks for data modeling and an implementation of the framework are discussed.

1. Introduction

A data model is a representation of the state space and transition behaviour of the system under consideration: the object system or Universe of Discourse (UoD). The object system is that part of the real world that we are interested in. A data model is, on one hand, a description of the object system; on the other hand it is an abstract specification of a database system to be used to monitor or supply information about the object system.

There are many frameworks to represent state spaces for object systems, for instance: relational models [COD70], network models [BAC69], entity relationship models [CHE80], functional models [SHI81], binary models [ABR74], [NIJ77], IFO- [ABI87], and NF^2-models [SCH83]. There are many variants of these models, therefore we use plural form. Note that the term *models* is used here in the sense of meta model: a model to describe models. We prefer the term *framework* for meta models.

All frameworks mentioned above only enable the designer of a database to construct a state space that is in general too large: the designer has to define some Boolean function over the state space to restrict the set of states to the feasible state space or database universe. The possibilities to define these constraints are in most frameworks rather poor. The relational model for instance considers only dependencies. Many frameworks have a query language and some of them a language for updates.

The construction of a data model for an object system proceeds along the following lines: (1) establish the boundaries of the object system; (2) specify a so-called free state space; (3) specify constraints to restrict the free state space to the feasible state space; (4) specify a set of queries that covers the information needs; (5) specify events occurring in the object system that cause state changes and therefore updates in the database. The ordening of steps is not strict.

Analysing the design process of a data model we find the following requirements for a data modeling framework:

1. there should be a language for formulating specifications 1 to 5 above,
2. the framework should be formal: a data model is a formal specification,
3. there should be a diagram technique for graphically representing essential parts of the design, in particular for the benefit of users who don't understand formal specifications,
4. it should be rather easy to express real world aspects in a data model.

Many of these requirements are rather vague; however, they provide a basis to compare frameworks. First of all most frameworks do not satisfy 1 and 2. The relational model does not have a diagram technique and is poor with respect to point 4. Several modern frameworks, such as the IFO- and NF^2-models, emphasize facilities for expressing complex objects. Most frameworks support the view that the real world consists of objects belonging to different types that are related to each other.

The framework we present is related to functional and binary data models as well as to entity relationship models. It has only a few basic concepts, a complete datalanguage that is based on first order logic and a diagram technique. When we would have to classify our framework, we would call it a functional data model. Functional data models have been studied before in the literature, most notably by David Shipman [SHI81]. In his 1981 paper Shipman builds his model around the basic concepts of <u>entities</u> (the "things that exist" in the object system) and <u>functions</u> (relations between entities). The essential difference between Shipman's data model and ours is that Shipman's functions are multivalued in the sense that when applied to an entity in its domain a function always will yield a set of entities. In our case single entities will be returned. Another difference is that Shipman also treats datatypes such as strings and integers, needed for representing names and numbers, as entities. In our model such objects do not appear at the data model level (except perhaps in some very special cases such as the modeling of a programming language), but only occur when we discuss representational issues.

We have been using the framework already for several years in data modeling courses and in practice. It is easy to transform a scheme in our framework into a relational scheme (see section 6). However, we have built a database management system that directly translates schemes from our framework into data structures (see also section 6).

In section 2 we define schemes and the construction of a free state space from a scheme. In section 3 we present our data language: the syntax, in an informal way, and the semantics. In section 4 we consider a diagram technique. In section 5 we introduce extended schemes and discuss their impact. Finally, in section 6, we give an example and some comparisons to other data models. We make some comments and mention some research topics.

2. Scheme and free state space

In the world view of our framework there are objects, having properties. Objects are organised into categories. Objects having similar properties will belong to the same category. In each state a category consists of a finite number of different objects. In each state a property of a category is a function from that category to another or the same category. So properties are functional relations between categories.

To each category belongs a set that is called the domain of that category. It contains the representations of all objects that can possibly be a member of that category.

The scheme of a data model describes the structure of the free state space, not what contents the database will have at some point in time. In other words, we consider a database as a variable and the free state space as its type. The first step in the construction of a data model is the specification of a scheme.

Definition 1. Scheme

A scheme is a 5-tuple $<C, P, D, R, V>$, where

- C is a finite set;
- P is a finite set;
- $P \cap C = \varnothing$;
- $D \in P \to C$;
- $R \in P \to C$;
- V is a set-valued function; dom(V)=C

□

An element $c \in C$ is called a category ; an element $p \in P$ is called a property. D is called the domain category function ; D(p) is the domain category of property p. R is the range category function ; R(p) is the range category of property p. We have : $\mathrm{rng}(R) \cup \mathrm{rng}(D) = C$. V is called the domain function. For $c \in C$, V(c) is called the domain of category c.

Every object is unique although the domains of two different categories may overlap. This means that objects of different categories may have the same representation, however, they are uniquely identified by the combination of their category name and representation.

At this point, and also in the first steps of the design process, one should not worry about the way objects in a category are represented. One could use integers, strings, tuples or whatever for representing these objects as long as the representation of each object within a category is unique.

A database scheme [see, e.g., BAC69] may be represented by a semantic network , i.e. a directed graph with labeled edges and nodes. For every category $c \in C$ there is exactly one node in the graph with label c. For every pair of nodes in the graph for categories c and c', such that there is a property p with D(p)=c and R(p)=c', there is an edge in the graph directed from node c to node c'. This edge is labeled with property p. All labels are unique, since $P \cap C = \varnothing$.

In a database scheme we express two things:

- the classification of the objects of the Universe of Discourse into categories.
- the functional relationships of objects in one category to objects from other categories (or the same category), indicated by the properties.

Example of a Scheme

As an example of a database scheme represented by a directed graph we give the semantic network for a fragment of a University database. This database contains facts about students and the courses they take. The central objects in the following model are therefore the objects "student" and "course". The relationship between students and courses, represented by the object "enrol(ment)" is many_to_many : students usually enrol in more than one course in a given period and usually more than one student is enrolled in a particular course in a given period. Of a student the name, address, date of birth and department are registered. When students have attended a course they usually want to take an exam for the course for which they get a grade. Exams for a particular course can only be taken a few times a year. Since the University offers many courses, numerous courses have to be examined on the same day, so the object "exam" represents an m-to-n relationship between the objects "course" and "date". Since many students may take a particular exam and a student normally will take a number of exams, there also exists an m-to-n relationship between the objects "exam" and "student". This relationship is represented by the category "test". Since "grade" is really a property of the test, it is made an attribute of test. Finally, apart from their own unique code, courses also have a name. The network corresponding to this short description is given below. Nodes are represented by boxes, edges by arcs.

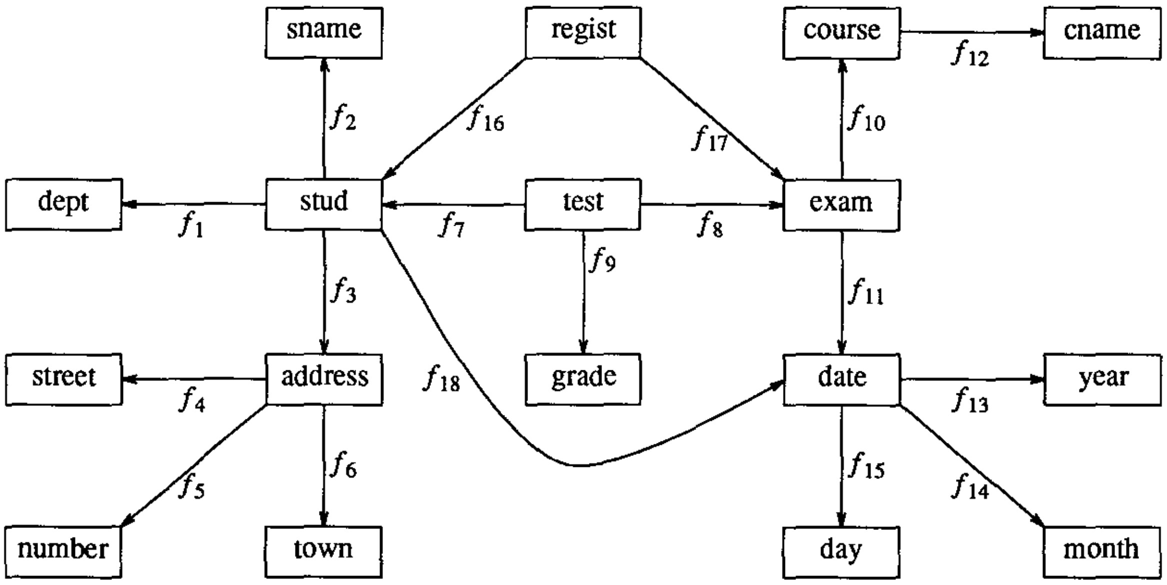


Fig.1 : Semantic Network for a University Database

We see from the graph that adresses are being specified in terms of a street, a (building)number, and a town and dates are being specified in terms of a year, a month, and a day.

A question raised frequently is why properties should be modeled by functions and not by binary relations? Our answer is that it suffices to have sets and functions to model object systems, whereas binary relationships are not primitive enough. When we want to model a relationship between two or more categories we define a new category that has one property for each of the related categories.

We will distinguish several types of categories. Similar distinctions have been made in the entity relationship model of Chen [CHE76] which has found a widespread usage (see, e.g., [CHE80], [CHE83] and [DAV83]; for an overview of data models, see [TSI82]). Since the ideas are rather similar we will use the same terminology, although the reader should be aware of the fact that the underlying mathematical notions are different. The first type is defined on the basis of scheme properties :

Definition 2. Attributes

Let $F = < C, P, D, R, V >$ be a database scheme. A category $c \in C$ is called an attribute category if and only if $c \notin \text{rng}(D)$

□

Hence attribute categories label nodes without outgoing edges. Therefore, these categories don't have properties of their own. Their role is to give further detail of the categories they are associated with through a property. In the example above, categories such as "year", "month", "day" and "cname" are attribute categories.

The second step in the construction is the definition of the free state space, also called the free universe of a database. A state will be defined as a set valued function with $C \cup P$ as domain.

Definition 3. Free State Space

Let $< C, P, D, R, V >$ be a database scheme. The free state space is the set of functions S^f with domain $C \cup P$ such that for $s \in S^f$:

i) $\forall c \in C : s(c) \subseteq V(c)$ and $s(c)$ is finite

ii) $\forall p \in P : s(p) \in s(D(p)) \nrightarrow s(R(p))$

□

Here $A \nrightarrow B$ stands for the set of all partial functions from A into B.

For a given state $s \in S^f$ of the database and a $c \in C$, $s(c)$ will be called the state of category c. Similarly, for a $p \in P$, $s(p)$ will be called the state of property p. Requirement i) then says that in each state only objects from the domain may belong to the state of the category. Furthermore, the number of objects in any state of category c is finite. ii) says that the state of a property is a partial function from the domain category to the range category. We see that $\text{dom}(s(p)) \subseteq s(D(p)) \subseteq V(D(p))$ and $\text{rng}(s(p)) \subseteq s(R(p)) \subseteq V(R(p))$.

3. Data language

The free state space is in general too large, i.e., it contains states that will or may not occur in the Universe of Discourse. In order to formulate restrictions on the state space and to express queries and updates, we define, in the third step of our construction, a first order language. Note that each database has its own language, which differs from the language of other databases only in the constants of the language. Our definition proceeds in the standard way [LEW81, LLO84].

Before we give the definition of the data language we introduce for later convenience some notation. The inverse of a function $f \in A \to B$ is defined by means of the inverse function application symbol "$^\vee$":

$$\forall b \in B : f^\vee b = \{ a \in A \mid (a; b) \in f \}.$$

$f^\vee b$ is also referred to as the complete original of b under f. Given the notions of function and inverse function application, we need to specify the application of a given function to a subset of its domain in order to be able to use composition of function and inverse functions. We first introduce for a set of sets W the generalised (or distributed) union $\cup$, which we define as :

$$\cup W = \{ x \mid \exists A \in W : x \in A \}.$$

$\cup W$ then is the set containing all elements that occur in at least one of the elements of W. We then are able to define for $f \in A \not\to B$, $D \subseteq A$ and $E \subseteq B$:

- $f \cdot D := \{ e \in \mathrm{rng}(f) \mid \exists d \in D : (d; e) \in f \}$,
- $f^\vee E := \cup\{ f^\vee e \mid e \in E \}$

The composition of functions $f \in A \to B$ and $g \in B \to C$ is denoted as $g \circ f$ ($g \circ f \in A \to C$). The restriction operator is denoted as $\restriction$ and the symmetric difference operator as $\div$. Finally, we write $I\!D = \cup\{ V(c) \mid c \in C \}$. $I\!D$ contains the elements of all domains. We assume that each separate domain has a total ordering. This is the only assumption we make for the domains.

Definition 4. First Order Language

Let $F = < C, P, D, R, V >$ be a database scheme. Its first order language (FOL) L_F then consists of the following elements :

i) Alphabet

The alphabet is the union of the following sets of symbols

- constants : $I\!D \cup C \cup P$
- variables : $\{ X, Y, Z, X_1, Y_1, Z_1, \ldots \}$
- function symbols : $\{ \circ, \cdot, ^\vee, \mathrm{dom}, \mathrm{rng}, \restriction \}$
- set symbols : $\{ \cup, \cap, \setminus, \div \}$
- atom comparison symbols : $\{ =, \leq, \geq \}$
- set comparison symbols : $\{ \subseteq, \supseteq, = \}$
- function comparison symbols : $\{ \subseteq, \supseteq, = \}$

- atom-set symbols : $\{ \in \}$
- logical symbols : $\{ \wedge, \vee, \neg, \rightarrow, \leftrightarrow \}$
- quantors : $\{ \underline{A}, \underline{E}, \underline{S} \}$
- interpunction symbols : { [,], {, }, (,), |, :, ;, , }

(We note that symbols such as = and $\subseteq$ are being overloaded : they can be used in more than one context.)

ii) Terms

- every $a \in \mathbb{D}$ is an a-term
- every variable is an a-term
- every $c \in C$ is an s-term
- every $f \in P$ is an f-term
- if $a_1, \ldots, a_n$ $(n \in \{1, 2, \ldots\})$ are a-terms then $\{a_1, \ldots, a_n\}$ is an (enumerated) s-term
- if $a_1, \ldots, a_n, b_1, \ldots, b_n$ are a-terms then $\{(a_1; b_1), \ldots, (a_n; b_n)\}$ is an (enumerated) f-term
- a-, s- and f-terms are terms
- if f and g are f-terms, then $f \circ g$ is an f-term
- if f is an f-term and x is an a-term, then $f \cdot x$ is an a-term and $f^{\vee} x$ is an s-term
- if f is an f-term, then dom(f) and rng(f) are s-terms
- if f is an f-term and s an s-term, then $f \restriction s$ is an f-term
- if f is an f-term and s an s-term, then $f \cdot s$ and $f^{\vee} s$ are s-terms
- if s_1 and s_2 are s-terms and θ is a set symbol then $s_1 \theta s_2$ is an s-term
- if X is a variable, $c \in C$ and q a predicate, then $\underline{S}[\, X : c \mid q \,]$ is an s-term
- there are no other terms

iii) Predicates

- if a_1 and a_2 are a-terms and θ is an atom comparison symbol then $a_1 \theta a_2$ is a predicate
- if a is an a-term, s an s-term and θ a atom-set symbol, then $a\theta s$ is a predicate
- if s_1 and s_2 are s-terms and θ a set comparison symbol, then $s_1 \theta s_2$ is a predicate
- if f_1 and f_2 are f-terms and θ a function comparison symbol, then $f_1 \theta f_2$ is a predicate
- if q_1 and q_2 are predicates and θ is a logical symbol, not equal to $\neg$, then $(q_1 \theta q_2)$ is a predicate
- if q is a predicate, then $\neg q$ is a predicate
- if X is a variable, q a predicate and $c \in C$ then $\underline{A}[\, X : c \mid q \,]$ and $\underline{E}[\, X : c \mid q \,]$ are predicates

- there are no other predicates

□

Note that our language has some constructs that do not contribute to its expressive power, for instance the S̲-quantor and ˇ are superfluous. However, they are useful for the user because they are well known mathematical constructs and allow for compact specification of queries and constraints.

As an example of an element of L_F, consider the following statement, based on the database scheme of Fig.1: "Every exam has a course and a date associated with it". This statement is expressed in L_F as:

$\underline{A}$[e : exam | $\underline{E}$[c : course | f_{10}·e = c] ∧ $\underline{E}$[d : date | f_{11}·e = d]]

Given the alphabet the terms and predicates are being defined inductively. In L_F, a quantor binds a variable to an s-term (the domain of the variable). We employ the usual rules (LLO84) for the scope of such a binding to be able to distinguish between bound and free occurrences of a variable.

There are several context conditions, for instance if f ∈ P and a ∈ $\mathbb{D}$ then f·a is only defined if a ∈ V(D(f)) and if f·X is a term with X bound by a quantor over a category c then it has to be that V(D(f)) = V(c). The interpretation of terms and predicates is intuitively clear. However, we will give a formal definition of the interpretation of terms and predicates without free variables.

An interpretation of the first order language defined above is constructed according to the standard way (cf. [LEW81, LLO84]). Our interpretation function $\mathbb{I}$ will depend on the state of the database, given by s ∈ S^f. This dependence will be indicated by subscripting $\mathbb{I}$ with s : $\mathbb{I}_s$.

For symbols from the alphabet, other than the constants and variables, we will denote the interpretation using the underscored version of the symbol : if θ is such a symbol, then $\underline{\theta}$ stands for $\mathbb{I}_s(\theta)$. θ will have its usual mathematical meaning, e.g., if θ is ∘, $\underline{\circ}$ stands for the function composition operator defined above.

We will use the following notation for substitution : let p be a term or a predicate, then p_y^X denotes the term or predicate where each free occurrence of X is replaced by y. $\mathbb{P}$W, for a set W, denotes the powerset of W, i.e. the set of all subsets of W, include ∅ and W.

Definition 5. Interpretation

Let F = < C, P, D, R, V > be a database scheme with data language L_F. Let L_T be the set of terms without free variables and L_C the set of closed predicates. Furthermore, let S^f be the free universe of this database scheme. The interpretation function $\mathbb{I}$ satisfies:

- $\mathbb{I} \in S^f \times (L_T \cup L_C) \rightarrow \mathbb{D} \cup \mathbb{P}(\mathbb{D}) \cup (\mathbb{D} \rightarrow \mathbb{D}) \cup$ {true, ∗, false}, where ∗ is the truth-value for undefined.
- for x ∈ $\mathbb{D}$: $\mathbb{I}_s(x) = x$
- for $x_1, \ldots, x_n \in \mathbb{D}$: $\mathbb{I}_s(\{ x_1, \ldots, x_n \}) = \underline{\{} x_1 \underline{,} \cdots \underline{,} x_n \underline{\}}$

- for $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{D}$:
  $\mathbb{I}_s(\{(x_1; y_1), \ldots, (x_n; y_n)\}) = \underline{\{}\underline{(}\, x_1 \,\underline{;}\, y_1 \,\underline{)}\underline{,} \cdots \underline{,}\underline{(}\, x_n \,\underline{;}\, y_n \,\underline{)}\underline{\}}$
- for $x \in C$: $\mathbb{I}_s(x) = s(x)$
- for $x \in P$: $\mathbb{I}_s(x) = s(x)$
- for f and g f-terms : $\mathbb{I}_s(f \circ g) = \mathbb{I}_s(f) \,\underline{\circ}\, \mathbb{I}_s(g)$
- for f an f-term, a an a-term and $\theta \in \{ \cdot, \check{} \}$: $\mathbb{I}_s(f\theta a) = \mathbb{I}_s(f) \,\underline{\theta}\, \mathbb{I}_s(a)$
- for f an f-term and $\theta \in \{ \text{dom}, \text{rng} \}$: $\mathbb{I}_s(\theta(f)) = \underline{\theta}\, (\mathbb{I}_s(f))$
- for f an f-term, σ an s-term and $\theta \in \{ \restriction, \check{}, \cdot \}$: $\mathbb{I}_s(f\theta\sigma) = \mathbb{I}_s(f) \,\underline{\theta}\, \mathbb{I}_s(\sigma)$
- for σ_1 and σ_2 s-terms and $\theta \in \{ \cup, \cap, \setminus\ \div \}$: $\mathbb{I}_s(\sigma_1 \theta \sigma_2) = \mathbb{I}_s(\sigma_1) \,\underline{\theta}\, \mathbb{I}_s(\sigma_2)$
- for X a variable, $c \in C$ and q a predicate : $\mathbb{I}_s(\underline{S}\,[\, X : c \mid q\,]) = \underline{\{}\, y \,\underline{\in}\, \mathbb{I}_s(c) \,\underline{|}\, \mathbb{I}_s(q^{X}_{y})=\text{true} \,\underline{\}}$
- for a_1 and a_2 a-terms and $\theta \in \{ =, \leq, \geq \}$: $\mathbb{I}_s(a_1 \theta a_2) = \mathbb{I}_s(a_1) \,\underline{\theta}\, \mathbb{I}_s(a_2)$
- for σ an s-term and a an a-term : $\mathbb{I}_s(a \in \sigma) = \mathbb{I}_s(a) \,\underline{\in}\, \mathbb{I}_s(\sigma)$
- for f_1 and f_2 f-terms and $\theta \in \{ \subseteq, \supseteq, = \}$: $\mathbb{I}_s(f_1 \theta f_2) = \mathbb{I}_s(f_1) \,\underline{\theta}\, \mathbb{I}_s(f_2)$
- for σ_1 and σ_2 s-terms and $\theta \in \{ \subseteq, \supseteq, = \}$: $\mathbb{I}_s(\sigma_1 \theta \sigma_2) = \mathbb{I}_s(\sigma_1) \,\underline{\theta}\, \mathbb{I}_s(\sigma_2)$
- for q_1 and q_2 predicates and $\theta \in \{ \wedge, \vee, \rightarrow, \leftrightarrow \}$: $\mathbb{I}_s(q_1 \theta q_2) = \mathbb{I}_s(q_1) \,\underline{\theta}\, \mathbb{I}_s(q_2)$
- for q a predicate : $\mathbb{I}_s(\neg q) = \underline{\neg}\, \mathbb{I}_s(q)$
- for X a variable, $c \in C$ and q a predicate : $\mathbb{I}_s(\underline{E}[\, X : c \mid q\,])$ = true if there exists a $y \in \mathbb{I}_s(c)$ such that $\mathbb{I}_s(q^{X}_{y})$ = true, false if for all $y \in \mathbb{I}_s(c)$, it holds that $\mathbb{I}_s(q^{X}_{y})$ = false, and $*$ otherwise.
- for X a variable, $c \in C$ and q a predicate : $\mathbb{I}_s(\underline{A}[\, X : c \mid q\,])$ = true if for all $y \in \mathbb{I}_s(c)$ it holds that $\mathbb{I}_s(q^{X}_{y})$ = true, false if there exists a $y \in \mathbb{I}_s(c)$, such that $\mathbb{I}_s(q^{X}_{y})$ = false, and $*$ otherwise.

□

Note that it might happen that an expression is undefined, for instance by applying a function to an argument outside its domain. Then any comparison to this expression will also get the value $*$. We use the logic of partial functions (see [JON86]), hence we have a three-valued logic. In set definitions we only accept elements if the truth-value is true.

The Closed World Assumption is implicit in our definition of the interpretation function. When it does not follow from the state s of the database that, e.g., $x \in a$ for some set a and element x, then it is implied that $x \notin a$.

It is straightforward to verify by induction on the number of quantors that every term and predicate without free occurrences of variables has an interpretation according to the rules specified in definition 5.

It also follows directly from definition 5 that for each s-term st without free variables there is a category c such that $\forall s \in S^f : \mathbb{I}_s(st) \subseteq \mathbb{I}_s(c) = s(c)$. Hence each interpretation of an s-term without free variables equals a finite set.

The data language L_F does not recognize any structure in the elements of $\mathbb{D}$. Therefore, operations such as projection cannot be expressed in it.

Now we are able to define our datalanguage. It is an extension of the first order language. We first extend the alphabet with the quantor symbol ?, the operator symbols $\uparrow$ and $\downarrow$ denote insertion and deletion, respectively, and a connective symbol ; to indicate composition of two updates. We first give the syntax and afterwards the interpretation.

Definition 6. Data language

Let F = <C, P, D, R, V> be a scheme and let L_F be the first order language, with alphabet extended with the symbols ?, $\uparrow$, $\downarrow$ and ;. The data language L_D contains L_F and the following elements

(i) constraints

Every predicate in L_F, without free variables is a constraint. L_C denotes the sublanguage of L_F containing only constraints.

(ii) queries

Let $X_1, ..., X_n$ be distinct variables and let $c_1, ..., c_n$ be elements of C and q a predicate with at most $X_1, ..., X_n$ as free variables then

$$?[\, X_1 : c_1, ..., X_n : c_n \mid q \,]$$

is a query. L_Q is the sublanguage of L_D containing only queries.

(iii) updates

Let $c \in C$, $f \in P$, and let σ, σ' be s-terms and ϕ, ϕ' be f-terms without free variables, such that σ and ϕ are both enumerated, then:

$$c\uparrow\sigma,\ c\downarrow\sigma',\ f\uparrow\phi,\ f\downarrow\phi'$$

are updates. If u_1 and u_2 are updates then $u_1;u_2$ is also an update. L_U is the sublanguage containing only update expressions of the form above.

□

Note that in 6 iii) σ and σ' represent different kinds of information. σ represents, in general, information not yet present in the database state and thus (this is a restriction!) can only be specified by explicit enumeration. σ' on the other hand represents information to be removed and therefore can be specified by an expression from L_F. A similar restriction holds for ϕ with respect to ϕ'.

The interpretation of constraints is clear by definition 5. The interpretation of a query is also straightforward: it is a subset of a Cartesian product. If a query contains only one variable the query has the same interpretation as an s-term:

$$\mathbb{I}_s(\, ?[\, X : c \mid q \,]) = \mathbb{I}_s(\, \underline{S}[\, X : c \mid q \,]).$$

However, since we want the datalanguage for our model to be as expressive as tuple calculus or relational algebra are for the relational models, we also allow the formation of pairs or, more general, n-tuples of objects.

The interpretation of the updates is more complicated. The choice of the elementary updates is motivated by the fact that they allow for deletions from and insertions into categories and properties such that the result will be part of the free state space. Other constraints are not taken into account. They have to be dealt with in a layer above the datalanguage.

<u>Definition 7. Interpretation of the data language</u>

Let F = <C, P, D, R, V> be a scheme with data language L_D. The interpretation function $I\!I$ defined in definition 5 is extended by taking the union with two other functions: $I\!I' \in S^f \times L_Q \rightarrow \bigcup_{n \in I\!N} I\!P(I\!D^n)$ and $I\!I'' \in S^f \times L_U \rightarrow S^f$. Let s, s' $\in S^f$ and $I = I\!I \cup I\!I' \cup I\!I''$.

(i) Let $X_1, ..., X_n$ be distinct variables, $c_1, ..., c_n \in C$ and q a predicate with at most $X_1, ..., X_n$ as free variables, then

$$I_s(\,?[\,X_1{:}c_1, ..., X_n{:}\; c_n \mid q\,]) = \{(y_1, ..., y_n) \in I_s(c_1) \times ... \times I_s(c_n) \mid I_s(q^{X_1 \cdots X_n}_{y_1 \cdots y_n}) = \text{true}\}$$

(ii) Let $c \in C$ and $a_1, ..., a_n \in I\!D$ then $I_s(c \uparrow \{a_1, ..., a_n\}) = s'$ such that for all $x \in C \cup P$:
$x \neq c \rightarrow s'(x) = s(x)$ and
$s'(c) = s(c) \cup (\{a_1, ..., a_n\} \cap V(c))$

(iii) Let $f \in P$ and $a_1, ..., a_n, b_1, ..., b_n \in I\!D$ such that $a_1, ..., a_n$ are distinct. Let further $\{(c_1; d_1), ..., (c_m; d_m)\}$ be the set $\{(a_1; b_1), ..., (a_n; b_n)\} \cap V(D(f)) \times V(R(f))$. Then
$I_s(f\uparrow \{(a_1; b_1), ..., (a_n; b_n)\}) = I_s(f\uparrow \{(c_1; d_1), ..., (c_m; d_m)\} = s'$ for $m \leq n$, such that for all $x \in C \cup P$:
$x \neq f \wedge x \neq D(f) \wedge x \neq R(f) \rightarrow s'(x) = x$, and
$s'(D(f)) = s(D(f)) \cup \{c_1, ..., c_m\}$
$s'(R(f)) = s(R(f)) \cup \{d_1, ..., d_m\}$
$s'(f) = (s(f) \setminus \{\, (x; y) \in s(f) \mid x{=}c_1 \vee ... \vee x{=}c_m \,\}) \cup \{(c_1; d_1), ..., (c_m; d_m)\}$

(iv) Let $c \in C$, X a variable and q a predicate with at most X as free variable, and let $Rc = \{\, y \mid y \in I_s(\,\underline{S}[\,X : c \mid q\,]) \wedge (\forall f \in R^{\vee} c \mid y \notin I_s(rng(f)))\}$ then
$I_s(c \downarrow \underline{S}[\,X : c \mid q\,]) = s'$ such that for $y \in C \cup P : y \neq c \wedge y \notin D^{\vee} c \rightarrow s'(y) = s(y)$ and
$s'(c) = s(c) \setminus Rc$
$s'(f) = s(f) \setminus \{\, (x; y) \mid x \in Rc \,\}$ for all $f \in D^{\vee} c$

(v) Let $f \in P$, X be a variable and q a predicate with at most X as free variable, then
$I_s(\, f \downarrow \underline{S}[\,X : c \mid q\,]) = s'$, such that for all $y \in C \cup P : y \neq f \rightarrow s'(y) = s(y)$ and
$s'(f) = s(f) \setminus \{\, (x; z) \in s(f) \mid x \in I_s(\,\underline{S}[\,X : c \mid q\,])\}$, provided that $c = D(f)$, otherwise $s' = s$.

(vi) Let u_1 and u_2 be updates then $I_s(u_1;u_2) = I_{s'}(u_2)$, where $s' = I_s(u_1)$

$\square$

It is easy to verify that for all updates u : $I_s(u) \in S^f$. The insert operator $\uparrow$ when applied to a category only has a local effect (ii). Application of the insert-operator $\uparrow$ to a property (iii) means in fact a modification since old values are deleted. Moreover, objects that are not yet present in the domain or range category, are inserted. When the deletion operator $\downarrow$ is applied to a category (iv), only those objects are deleted that do not occur in the range of an incoming property. For those objects also the outgoing properties are updated. The application of the deletion operator $\downarrow$ to a property only has a local effect. The updates define transitions on the free state space. They belong to events in the universe of discourse. Next we will restrict the free state space by constraints.

Definition 8. Feasible state space

Let F = <C, P, D, R, V> a schema and SoC be a set of constraints then the feasible state space S is: $S = \{ s \in S^f \mid \underline{A}[\, q : SoC \mid I_s(q) = true \,] \}$, where I is the interpretation induced by F

□

It is required that updates keep the constraints invariant. At the level of data modeling it is sufficient to require this. At the implementation level the update programs should be verified against these constraints. The following property is obvious. We will use it in section 5.

Lemma 1

Let $c \in C$, $a_1, ..., a_n \in \mathbb{D}$, $s \in S^f$ and $k \in \{ 1, ..., n \}$, then

$$I_s(c\uparrow\{a_1, ..., a_n\}) = I_s(c\uparrow\{a_1, ...,a_k\};\ c\uparrow\{a_{k+1}, ..., a_n\})$$

□

4. Diagram technique and Constraints

As stated before, a database scheme may be represented by a directed graph with labeled edges. Although this graph gives a complete representation of the basic structure of the database, it does not allow one to express any semantic information that may be available. In this section we will introduce a diagram technique [see BAC69] which will give us an overview of the most important aspects of the data model, starting from the graphic representation of the database scheme.

Diagrams

As a first step we will introduce special symbols for the various types of categories and properties that we have encountered so far. We start from the scheme and replace every node with the symbol which is appropriate for the kind of category it represents. First, an attribute category will be represented by a circle, other categories by a rectangle. Each symbol will contain the category name. Secondly, every edge stands for a property and will be represented by a continuous line with an arrow in the direction of the corresponding edge. When the situation is clear from the diagram : for instance, when a property has an attribute category for its range, the arrow will be dropped from the line. All lines will be labeled with the corresponding property name.

Standard Constraints

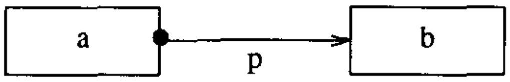
Secondly, we will include special symbols and we'll give the translation of these diagram conventions into elements of L_F. In the following, we will use 'category a' or 'a' to denote the state s(a) of the category with name a, given a database state s. Similarly, we will use 'property p' or 'p' to denote the state s(p) of a property with name p. We distinguish the following cases :

Completeness Constraint

Often a property $p \in P$ with domain category a and range category b is required to be complete, i.e., it is required to satisfy :

dom(p) = a.

A property satisfying this constraint is drawn in the diagram with a solid dot at the base of the property line:



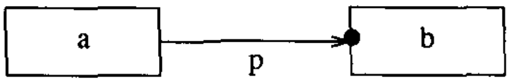
The imposition of a completeness requirement on a property is one way of modeling referential integrity. The property then, obviously, represents the fact that for every element of its domain there exists an element in its range. For instance, the date of every exam should be a valid date (falling in one of the reserved periods), and the subject of every exam should be an existing course.

Onto Constraint

A property $p \in P$ with D(p) = a and R(p) = b is said to be from a onto b or surjective when :

rng(p) = b.

A property p satisfying this constraint has a solid dot at the tip of the property arrow :



Typically, properties with an attribute category as range category will satisfy this surjectivity constraint as it is alternatively called. When a given attribute category b serves as range category for a number of properties, it may happen that none of them separately is onto b, but that all of them together are. These properties then satisfy a combined surjectivity constraint. For properties $f \in a \rightarrow b$ and $g \in c \rightarrow b$, such a constraint would read :

$\underline{A}[\ y : b \mid y \in rng(f) \vee y \in rng(g)\ ]]$

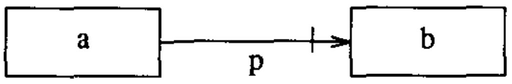
A property that isn't onto, is into.

One-to-one Constraint

When a property $p \in P$ with $R(p) = b$ and $D(p) = a$ is <u>one-to-one</u> from a to b it satisfies :

$$\underline{A}[\ x : a \mid \underline{A}[\ y : a \mid (x \in dom(p) \wedge y \in dom(p)) \rightarrow (p \cdot x = p \cdot y \rightarrow x = y)\ ]]$$

A property p satisfying a one_to_one constraint appears in the diagram with a cross bar on the property line:



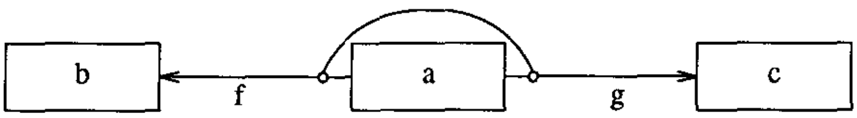
A property p that is one-to-one from one category to another and also complete, but not onto can be used to model <u>specialisation</u> (the ISA relationship in the Entity-Relationship model of Chen [CHE76]). An object O in category a then has a unique connection with an object O' from category b and through property p it inherits all the properties of O'. This construction is particularly useful when the objects of category a have all the properties of the objects of category b, but in addition have some properties of their own that are not shared by all objects in category b. As an example consider a category "personnel" of the University, which has properties like "name", "address" and so on. However, some employees belong to the teaching staff and give certain courses, while other employees belong to the administrative staff and are specialised in financial or insurance matters.

<u>Key Constraint</u>

Consider a set A of properties with the same domain category a : $A \subseteq D^{\vee} a$. For instance, let $A = \{f, g\}$ with $f \in a \rightarrow b$ and $g \in a \rightarrow c$. We call these properties <u>key properties</u> of a when the following key constraint is satisfied :

$$\underline{A}[\ x : a \mid \underline{A}[\ y : a \mid (\ x \in dom(f) \wedge x \in dom(g) \wedge y \in dom(f) \wedge y \in dom(g)) \rightarrow ((\ f \cdot x = f \cdot y \wedge g \cdot x = g \cdot y\ ) \rightarrow x = y\ )]]$$

A key constraint is denoted by an arc between the participating properties. Since these properties do not have to correspond to adjacent lines in the diagram, the arc has a symbol "o" on the appropriate property lines. A single category may have more than one set of key properties. Only minimal keys are indicated in the diagram. A set of key properties is minimal when omission of any one of the properties from the set yields a set of properties that does not satisfy the key constraint. Consider for example the case that the properties f from a to b and g from a to c are required to be key properties of category a. This is expressed in a diagram as :



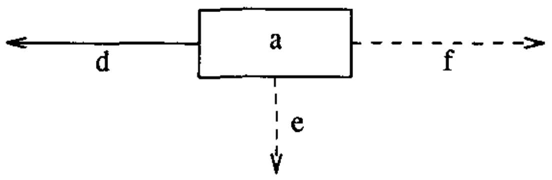
Please note, that every element in a category has a unique identification. An important difference between this intrinsic identification and identification through a set of key properties is that the intrinsic identification has to be present, otherwise the object does not exist. Key properties only provide unique identification for an object in their domain category, when the complete set of objects in the range categories for the object is known. We'll return to identification issue in more detail when we discuss representations.

When the set A of properties is a singleton, we recover, as a special case of the key constraint, the one-to-one constraint!

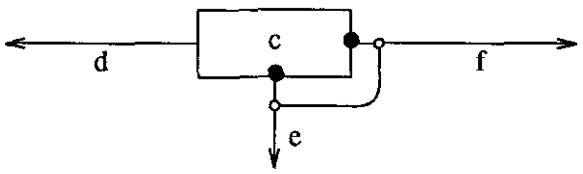
When a property p with domain D(p) = a and range R(p) = b is constrained to be complete and one-to-one from a to b, it can be considered as modeling a one-attribute-key constraint. When this property moreover is required to be onto as well the two categories become equivalent : every element in a then correponds to precisely one, unique element of b. For example, we can require every student in the database to have a unique student identification number.

Primary Keys

A set of key properties is called identifying a category c when the set forms a minimal key for c and all properties in the set are complete. In this case we can associate with every object in category c a unique set of objects in other categories, which can be used to identify the object in c. When a set of properties is identifying, we draw the properties in the diagram with dashed lines:



This is shorthand for:



This convention allows us to indicate only one set of identifying properties. These properties (e and f) form the so-called primary key of the category (c).

5. Extended scheme and representations

At this point we have seen two different ways of identifying an object in a category. First of all, every object is unique and therefore can be uniquely identified. But then also, some categories are required to satisfy a primary key constraint. Consequently, the objects in such a category can be identified by means of a set of objects in other categories. In addition to their intrinsic identification, they also have an external identification.

We therefore sometimes have a choice between two options. We can base the representation of the objects in a category on the intrinsic identification of the objects. Categories with such a representation are called basic categories. The subset of C containing the names of these categories is called CB. This option typically is the (only) one for attribute categories.

Or, in case of a category satisfying a primary key constraint, one has the alternative of composing the representation of such a category from the key properties and the representations of the range categories of these key properties. Categories with such a representation are called derived categories. We denote the subset of C containing the names of these categories with CD. Obviously $C = CB \cup CD$ and $CB \cap CD = \varnothing$.

We introduce the concept of an extended scheme. Such a scheme can be used to construct an ordinary scheme and some primary key constraints. Hence on the one hand it provides us with a more compact way to describe data models possibly using diagrams, on the other hand it gives us the flexibility to derive representations for categories where no natural representation of objects is available (we have already encountered such object in the example database scheme, viz. "test", "exam", "date" and "address")

Definition 9. Extended scheme

An extended scheme is a 7-tuple <CB, CD, P, D, R, B, G> where

- CB, CD, P are mutually disjoint, finite sets,
- $D \in P \rightarrow (CB \cup CD)$,
- $R \in P \rightarrow (CB \cup CD)$,
- B is a set-valued function with dom(B) = CB
- $G \in CD \rightarrow \mathbb{P}(P)$ such that for all $c \in dom(G) : G(c) \subseteq D^{\vee} c$.

□

The elements of $CB \cup CD$ are categories, P, D and R have the same meaning as in definition 1 and B fullfils the role of domain function V for the basic. categories. The function G determines the properties of a derived category that form the primary key. For a category $C \in CD$ we will use the categories in $R(G(c)) = \{y \in CD \cup CB \mid \exists f \in G(c) : y = R(f)\}$ for the representation of objects of c. It is clear that we can't use a category d to represent c when the representation of d depends on c. Hence the derivation may not contain a cycle. In the next definition we formalise this concept.

Definition 10. Derivation relation

Let E = <CB, CD, P, D, R, B, G> be an extended scheme, then the derivation relation T satisfies:

- $T \subseteq dom(G) \times dom(G)$,
- $(x; y) \in T \leftrightarrow G(x) \cap R^{\vee} y \neq \varnothing$

□

The requirement that the derivation does not contain a cycle is equivalent with the requirement that the transitive closure T* of T is irreflexive. We call this the finite derivation property. Under this condition we can construct a scheme from an extended scheme. Before we do so, we first introduce the generalised product operator Π. Let P be a set-valued function, then

$$\Pi(P) = \{ p \mid p \text{ is a function with domain dom}(P) \text{ and } \forall x \in \text{dom}(p)\text{: } p(x) \in P(x) \}.$$

Lemma 2

Let E = <CB, CD, P, D, R, B, G> be an extended scheme with the finite derivation property. Then the 5-tuple <C, P, D, R, V> where

- $C = CB \cup CD$,
- for $c \in CB : V(c) = B(c)$,
- for $c \in CD : V(c) = \Pi(\lambda x \in G(c) : V(R(x)))$

forms a scheme.

Proof: It is easy to verify by induction that the recursive definition of V is sound due to the irreflexivity of T*. The rest is trivial.

□

We require that for each category $c \in CD$ the properties G(c) satisfy the primary key constraint. Further we will require that an extended scheme will satisfy the finite derivation property.

We see that a database scheme can be represented in very many ways, depending on the choice of B and G. Each choice of representation has its own emphasis and implications. Note, that we have not required that every category satisfying a primary key constraint is an element of dom(G). However, unless there is a good reason not to do so, it is better to avoid the redundance introduced by keeping the domain of a category satisfying a primary key constraint basic. At this point we can, on the basis of their properties, distinguish three types of categories. We already encountered the attribute categories (see def. 2), which are categories without properties. The other two types are given in definition 11.

Definition 11. Entities and Relationships

Let E = < CB, CD, P, D, R, B, G > be an extended scheme.

A $c \in C$ is called an entity category if and only if

- c is not an attribute category and
- $c \notin \text{dom}(G) \vee \forall f \in G(c) : R(f)$ is an attribute category.

A $c \in C$ is called a relationship category if and only if c is neither an attribute nor an entity category.

□

Note again that our definitions are very similar to most definitions of the entity relationship model. Relationship categories typically have a derived domain as have some entity categories such as "date" in fig. 1. An entity category will be represented by a box, a relationship category be a diamond.

Our data model is a very general and flexible model. In fact, the relational data model (as well as some other models) can be regarded as a special case, namely the one specified by the following constraints:

- every category occurring as the domain category of a property has a primary key,
- the range category of every property is an attribute category.

Thus there are no relationship categories in the relational data model, only entity categories. Sometimes an extra constraint is imposed: all properties (also the non-identifying ones) are complete. If this constraint is imposed as in Codd's original proposal [CO70], no incomplete information can be represented. Such a requirement may turn out quite inconvenient in practical situations, where information often becomes available in portions. In the our data model, where, in general, partial functions are allowed, no such problems arise.

When we take the scheme of the University database, depicted in Fig. 1, and extend it by imposing a number of constraints and by choosing some representations, the diagram of Fig. 2 may result:

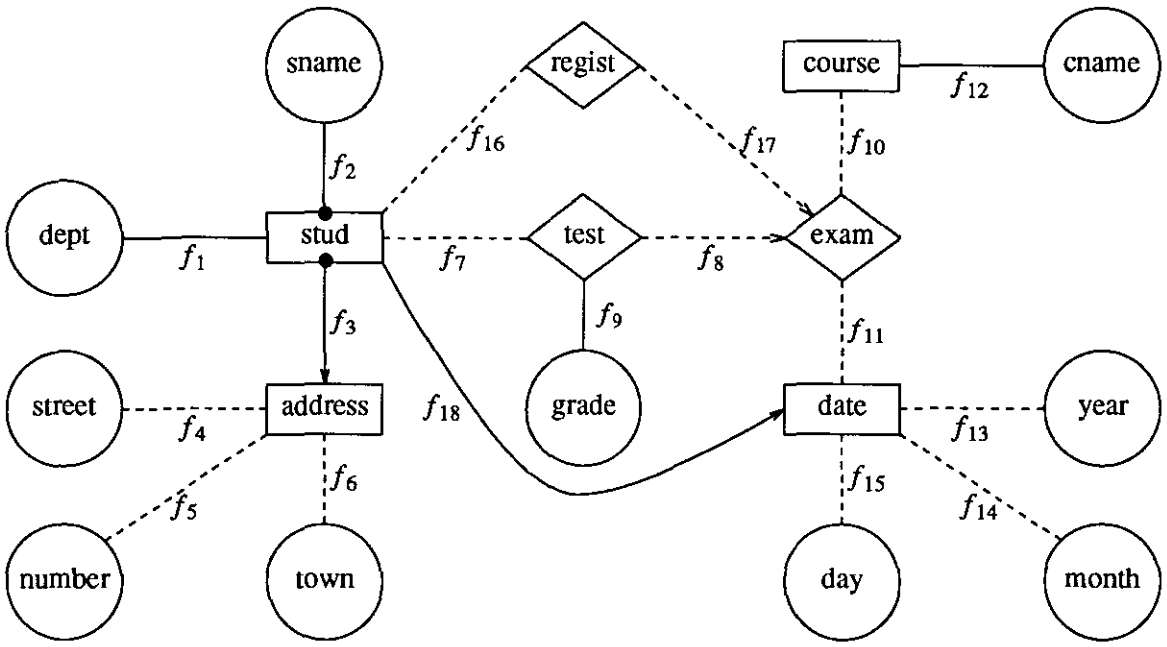


Fig.2 : Extended Scheme for a University Database

We immediately can identify which categories are attribute categories, and which ones are entity or relationship categories. All properties with an attribute category as range category moreover are subject of a surjectivity constraint. Inspection of Fig. 2 further tells us that the categories "address", "enrol", "test", "exam" and "date" have a derived domain whereas the other categories have a basic domain. Fig.2 expresses that to every test there corresponds a student and an exam, and that to every exam corresponds a course and a date. We also see that the name and address of every student is known. Students are represented by a unique registration number. We will not spell out the complete scheme but only point to a few illustrative features:

- the derived category enrol has f_{16} and f_{17} as primary key functions, therefore:

  (enrol; $\{ f_{16}, f_{17} \}$) $\in$ G and

  V(enrol) = $\Pi(\{(f_{16}$; V(stud)), (f_{17}; V(course))$\}$)

  where V(course) and V(stud) are basic domains, which are sets of suitable strings or integers,
- since we have denoted properties in Figs. 1 and 2 for reasons of clarity by simple labels such as f_{10}, an informal description of the meaning of the property may be useful, such as:

  f_{10}: the subject of the exam.

Since an extended scheme induces an ordinary scheme we don't need a new data language for extended schemes. Only the constants in the domains of categories in CD have a complex structure. Consider the next example:

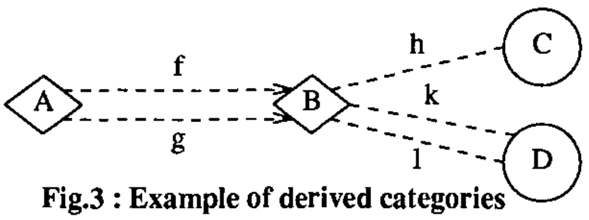


Fig.3 : Example of derived categories

An object a in A is represented as

$$a = \{ (f; \{ (h; c_1), (k; d_1), (l; d_2) \}), (g; \{ (h; c_2), (k; d_3), (l; d_4) \}) \}$$

Note that a itself is a function so that at the mathematical level the following, intuitively strange equation

$$f \cdot a = a \cdot f.$$

holds. Only the lefthandside is an expression in L_F.

In principle the semantics of the data language for the extended scheme is the same as for the induced ordinary scheme. Indeed, insertion and deletion operations on categories c $\in$ CB will have the same interpretation as before, and so do deletion operations on a derived category. However we will change the semantics of updates involving a derived category. The reason is that if, e.g., we add an element to category A in Fig. 3 then we want to update simultaneously the functions f and g and the category B. For B the same holds: we want to update the functions h, k and l and the categories C and D. The interpretation of an insertion or deletion operation on a property p $\in$ $\cup$rng(G) is that its state does not change. The state of these properties gets changed implicitely when the state of their domain gets changed. A deletion operation on a property p $\in$ P\$\cup$rng(G) is interpreted as before : it has only a local effect. The insertion operation on a property p $\in$ P\(R$^{\vee}$CD $\cup$ D$^{\vee}$CD) has the same interpretation as in Def.7, but the semantics of an insertion operation on a property p $\in$ (R$^{\vee}$CD $\cup$ D$^{\vee}$CD)\$\cup$rng(G) is changed in an obvious way, when it entails an insertion into a derived category.

In definition 13 we will formalise the concept of updates of derived categories. The insertion procedure sketched above will take us through intermediate states, which will in general not satisfy condition ii) of definition 3 : first a derived category and its primary key properties are updated, then, if necessary, the range categories of the key properties are updated; if any of these is a derived category the procedure is carried out once more, and so on. After the first round of updates it therefore is possible that the state of range of any of the key properties is not a subset of the state of the range category of that property. However, after completion of the update, the resulting state should again be an element of S. An alternative procedure would be to work backwards : first determine all basic categories affected, update these and then successively update all primary key properties and their derived domain categories. Let us first introduce

Definition 12 : Relaxed State Space
Let < C, P, D, R, V > be a database scheme. The relaxed state space is the set of functions S^r with domain $C \cup P$ such that for $s \in S^r$:

i) $\forall c \in C : s(c) \subseteq V(c)$ and s(c) is finite

ii) $\forall p \in P : s(p) \in V(D(p)) \not\twoheadrightarrow V(R(p))$

□

We note that the state of a property p now only depends on the domains of the domain and range categories, and no longer on their state.

Definition 13: Interpretation of updates in a derived category
Let E = <CB, CD, P, D, R, B, G> be an extended scheme with the finite derivation property. Let s be a state from the free state space of the scheme induced by E and let I be the interpretation from definition 7 with S^f replaced by S^r. Then the interpretation I' of E satisfies $I'_s(x) = I_s(x)$ for all expressions x except the following cases :

(i) for $c \in CD$ with $G(c) = \{ f_1, ..., f_n \}$ and $a \in V(c)$ we define :
$I'_s(c\uparrow\{a\}) = I'_{\tilde{s}}(R(f_1)\uparrow\{f_1 \cdot a\}; ...; R(f_n)\uparrow\{f_n \cdot a\})$, where $\tilde{s}$ is a state satisfying
$\tilde{s}(x) = s(x)$ for $x \in CB \cup CD \cup P \setminus \{c, f_1, ..., f_n\}$
$\tilde{s}(c) = s(c) \cup \{a\}$
$\tilde{s}(f_i) = s(f_i) \cup \{ (a; f_i \cdot a) \}$, for $i \in \{1, ..., n\}$

(ii) for $c \in CD$ and $\{a_1, ..., a_n\} \subseteq V(c)$:
$I'_s(c\uparrow\{a_1, ..., a_n\}) = I'_s(c\uparrow\{a_1\}; c\uparrow\{a_2\}; ...; c\uparrow\{a_n\})$

(iii) for $p \in \cup rng(G)$ and $a_1, ..., a_n, b_1, ..., b_n \in \mathbb{D}$ we define :
$I'_s(p\uparrow\{(a_1; b_1), ..., (a_n; b_n)\}) = s(p)$

(iv) for $p \in \cup rng(G)$, X a variable and q a predicate with at most X as free variable we define :
$I'_s(p\downarrow \underline{S}[X : c \mid q]) = s(p)$.

(v) Let $p \in (R^{\vee} CD \cup D^{\vee} CD) \setminus \cup rng(G)$ and $a_1, ..., a_n, b_1, ..., b_n \in \mathbb{D}$ such that $a_1, ..., a_n$ are distinct. Let further $\{(c_1; d_1), ..., (c_m; d_m)\}$ be the set $\{(a_1; b_1), ..., (a_n; b_n) \cap V(D(p)) \times V(R(p))$. Then
$I'_s(p\uparrow \{(a_1; b_1), ..., (a_n; b_n)\}) = I'_s(p\uparrow \{(c_1; d_1), ..., (c_m; d_m)\} = s'$ for $m \leq n$, such that for all

$x \in C \cup P$:

$x \neq p \wedge x \neq D(p) \wedge x \neq R(p) \rightarrow s'(x) = x$, and

$s'(D(p)) = I'_s(D(p) \uparrow \{c_1, ..., c_m\})$

$s'(R(p)) = I'_s(R(p) \uparrow \{d_1, ..., d_m\})$

$s'(p) = (s(p) \setminus \{ (x; y) \in s(p) \mid x=c_1 \vee ... \vee x=c_m \}) \cup \{(c_1; d_1), ..., (c_m; d_m)\}$

□

By lemma 1 the definitions are correct: the order of enumeration of elements of sets does not play a role.

6. Comments

In the course of the construction of a data model there are at every stage alternatives among which one has to choose. For instance, there is a lot of freedom in deciding what domain to choose for a category. Should the domain of a category be base or do the objects in this category derive their identity from their relationship to other categories, and should the domain therefore be derived? In fact, one could, given a database scheme, construct an equivalent database scheme from it with a domain specification in which only the attribute categories have a base domain (such as is the case in entity relationship models). If one would then arrange the model such that only attribute categories would occur as range category of a property, a data model would result with the structure of a relational data model [COD70].

At the representational level the data model defined above contains other data models such as the relational data models as special cases. This statement is corrobated by the fact that one can formulate an algorithm which generalizes, given a scheme, a relational representation for it.

When constructing complex representations, we have seen the beginning of a relational or tabular representation of our data model. Let's recall the definition of V(c) for $c \in dom(G)$:

$V(c) = \Pi(\lambda x \in G(c) : V(R(x)))$

This would be precisely the definition of a relation in a relational model which has attributes $x \in G(c)$. Thus, apart from the use of property labels as attribute names, objects from a category c with $c \in dom(G)$ are represented as tuples. It is not very hard to extend dom(G) in a systematic way to include not only all relationship, but also all entity categories. The construction of a relational representation proceeds along the following lines:

Algorithm

1) For every category, but those attribute categories which are the range of a set of properties satisfying a combined surjectivity constraint, introduce a relation with as name the name of the category.

2) Include in every relation one attribute, when the category with the same name has a base domain.

3) Include in every relation one attribute for every property whose range category has a base domain.

4) Include in every relation for every property, whose range category has a derived domain, as many attributes as the representation of that category requires.

5) Give every attribute a suitable name.

Step 4 really is an iterative step. When the range category of a property has a derived domain, we introduce in the first step as many attributes as the category has primary key properties. However, when any of these key properties has a range category with a derived domain, its attribute is replaced by as many attributes as that range category has primary key properties. And so on, until we arrive at a set of attributes corresponding to range categories with a base domain.

As an illustration of the algorithm we will construct a relational representation for the example of Fig.2. We will use the following notation:
relationname (attributename 1, .. , attributename k)
to represent the heading of a representation with name 'relationname' and k attributes. We get the following 7 relations:

stud (stud, sname, dept, street, number, town)

address (street, number, town)

test (stud, grade, course, year, month, day)

exam (course, year, month, day)

enrol (stud, course)

date (year, month, day)

course (course, cname)

We see, that a lot of information is duplicated in this simple version of the algorithm. The attributes corresponding to "address" and "date" are also included in the relations for "stud" and "exam" and "test" respectively. (In this simple example a straightforward refinement of the algorithm based on surjectivity constraints on f_3 and f_{11} would allow one to do away with the relations for "date" and "address"". This shows that there is no such thing as the relational representation.) By including the key attributes of a category with a derived domain in the representation all (hierarchical) structure is flattened out completely. This is a basic feature of the relational model, viz the use of flat ('normalized') relations. In fact, the representation produced by the algorithm above is in Boyce-Codd Normal Form for a database scheme subject to only standard constraints. In the relational model the association between attributes in different relations is made exclusively through the data language. In our data model this association is provided at the scheme level through the sharing of the same range category by several properties.

In the previous sections we have presented a formal framework for data modeling. The framework has been used for several years in courses and in practice. The concepts discussed above have to a large extent been implemented in a database management system called Eldorado.

The storage system of Eldorado realises the storage of the categories and properties in such a way that it involves only a limited amount of redundancy of data. Properties are implemented as sets of pairs of references to objects in categories and objects in derived categories are represented by internal identifiers. The database is stored in four files. There is an indexfile which has the structure of a B-tree-valued B-tree. At the first level one distinguishes between categories; at the second level between objects in the same category. As a consequence of this the objects in a category have an ordering. There is a reference file which stores the internal identifiers of the objects in a given database state together with lists of forward and backward references (for function- and inverse function application respectively) to the object. The location file records the position of the object in the objects file, which stores the actual representation of the object and an extension, which is an untyped amount of data, be it text, a picture or even a program code. This feature makes Eldorado suitable for supporting, e.g., Hypertext-applications.

The datalanguage discussed above has also been implemented in the Eldorado system, augmented with facilities to define schemes, do computations and format output (for the result sets of queries). We are investigating an extension of the datalanguage for defining and manipulating complex data objects, consisting for example of connected subgraphs of the diagram for the database scheme. With such language capabilities one would be able to treat, e.g., a student with his name and address or a course and all its exams as one object. This kind of structuring of objects is rather appealing from the point of view of both updating and querying the database. Another subject of study is the design of a graphical interface for defining and querying the database.

References

ABI87 Abiteboul, S. and Hull, R., IFO: A Formal Semantic Database Model, ACM TODS 12 (1987) 525-565.

ABR74 Abrial, J.R., Data semantics, in : Data Base Management, J.W. Klimbie and K.L. Koffeman, Eds, North Holland Pub. Co., Amsterdam (1974) 1-60.

BAC69 Bachman, C.W., Data Structure Diagrams, ACM SIGBDP Data Base 1, no 2 (1969) 4-10.

COD70 Codd, E.F., A Relational Model of Data for Large Shared Data Banks, Comm. ACM 13 (1970) 377-387.

CHE76 Chen, P.P., The Entity-Relationship Model -- Towards a Unified View of Data, ACM TODS 1 (1976) 9-36.

CHE80 Proceedings of the International Conference on Entity- Relationship Approach, Entity-Relationship Approach to System Analysis and Design, Los Angeles, 1979, North Holland Publ. Co. (1980), P.P. Chen, ed.

CHE83 Proceedings of the Second International Conference on Entity-Relationship Approach, Entity-Relationship Approach to Information Modeling and Analysis, Washington, 1981, North Holland Publ. Co. (1983), P.P. Chen, ed.

DAV83 Proceedings of the Third International Conference on Entity-Relationship Approach, Entity-Relationship Approach to Software Engineering, Anaheim, 1983, North Holland Publ. Co. (1983), C.G. Davis, S. Jajodia, B.-B Ng, R.T. Yeh, eds.

JON86 Jones, C.B., Systematic Software Development using VDM, Prentice Hall (1986)

LEW81 Lewis, H.R. and Papadimitriou, C.H., Elements of the Theory of Computation, Prentice Hall (1981)

LLO84 Lloyd, J.W., Foundations of Logic Programming, Springer- Verlag (1984)

NIJ77 Nijssen, G.M., Current Issues in Conceptual Schema Concepts, in Nijssen (Ed.), Architecture and Models in Data Base Management Systems, North Holland (1977)

SCH83 Schek, H.-J. and Scholl, M.H., The NF^2 Relational Algebra for a Uniform Manipulation of the External, Conceptual, and Internal Data Structures, in J.W. Schmidt (Ed.) Sprachen f:ur Datenbanken, IFB 72, Springer (1983)

SHI81 Shipman, D.W., The Functional Data Model and the Data Language DAPLEX, ACM TODS 6, (1981) 140-173

TSI82 Tsichritzis, D.C. and Lochovsky, F.H., Data Models, Prentice-Hall (1982)

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.

89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.