Towards a framework for comparing data models

Towards a Framework for Comparing Data Models

by

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

89/16

December, 1989

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

Towards a Framework for Comparing Data Models

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 proposed 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. Transformations to other frameworks for data modeling 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] (for an overview of data models, see also [TSI82]). 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.

Given the large amount of frameworks for data modeling it is a legitimate question to ask what these frameworks have in common and where they differ from each other? Moreover, anyone proposing a new framework for data modeling has to make clear what problems the framework solves, and what new possibilities it brings. In other words, one will have to compare with the other frameworks. In this paper we propose a framework for data modeling which is "upward compatible" with a great number of data modeling frameworks. It is based on the mathematical notions of set and function, and has a very general data language. It allows one to formally represent the constructs of a number of other data models, and then also to transform from one framework to another one, a feature which has wide applications in the context of distributed databases, where the various databases are based on different data 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 data 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 data language that is based on first order logic and a diagram technique. When we would have to classify our framework, we would call it an irreducible functional data model (see also section 6 for a more detailed discussion).

In section 2 we define schemes and the construction of a free state space from a scheme and give an example of a database 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 discuss representations and elaborate on the example. Finally, in section 6, we discuss the relation of our framework to the Entity-Relationship model and the Functional Data Model and give a transformation to a relational data model. 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 $\langle C, P, D, R, V\rangle$, where

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

□

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. 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 may be represented by a semantic network, i.e. a directed graph with labeled edges and nodes [see, e.g., BAC69], which represents our knowledge about the structure of the object system. 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 $P \cap C = \emptyset$.

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 their progress through college. Central objects in the following model are therefore the objects "student" and "exam". Students have a name, an address, and a date of birth, and they join in the course of their stay at the University a certain department. As a preparation for their degree they follow courses and take exams. Courses have, apart from their own unique code, a name. The University gives students several times a year the opportunity to take an exam for a given course. Since the University offers many courses, numerous courses have to be examined on the same day in parallel. "Exam" therefore represents a many-to-many relationship between the objects "course" and "date". Students have to register for a particular exam (again a many-to-many relationship, represented by the object "regist"), before they are allowed to complete the test for that exam. Since students normally will take a number of exams (on various occasions) and an exam usually is taken by many students the relationship between "student" and "exam"is n to m. It is represented by the object "test". Tests get graded. When the grade for a test is sufficiently high, the student has passed the corresponding exam. A network corresponding to this short description is given below. Nodes are represented by boxes, edges by arcs.

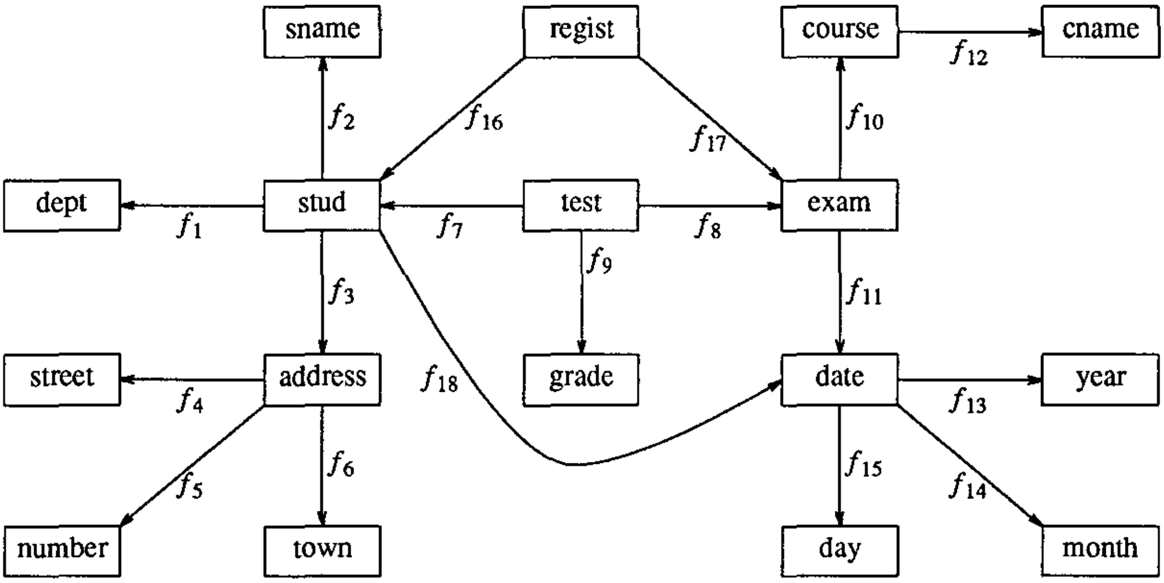


Fig.1 : Semantic Network for a University Database

We see from the graph that addresses 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.

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 2. 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 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 $dom(s(p)) \subseteq s(D(p)) \subseteq V(D(p))$ and $rng(s(p)) \subseteq s(R(p)) \subseteq V(R(p))$.

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].

Details of the construction can be found in [AER89]. The first order language L_F is defined for a database scheme F = < C, P, D, R, V >. It contains three elements : an alphabet, terms and predicates. The alphabet consists of a scheme-specific and a general part. The former contains the constants from the language : the category names, such as "test", "date" and "grade" and property names such as "f_1", and the elements of all domains : $\mathbb{D} = \cup\{V(c) \mid c \in C\}$, such as dates, coursenames, grades, and studentnames. The latter contains all symbols for constructing first order expressions on the basis of atomic terms such as the predicate X=3, which is constructed from the variable X and the constant 3, both atomic terms, and the comparison symbol =, set terms such as grade\{1,2,3} which is constructed from the set term grade (a category in the example) and the enumerated set {1,2,3} and the set symbol \, and predicates involving function terms such as

$f_9 \cdot X \geq 6$, where a function term f_9 is applied, using the function symbol $\cdot$, to a variable X and the result thereof is compared, using the comparison symbol $\geq$, to the a-term 6.

A more elaborate example of an element of L_F, based on the database scheme of Fig.1, expresses the statement : "Every exam has a course and a date associated with it" in the following way :

$$\underline{A}[\, e : exam \mid \underline{E}[\, c : course \mid f_{10} \cdot e = c] \wedge \underline{E}[\, d : date \mid f_{11} \cdot e = d]\,]$$

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.

The interpretation of elements of L_F also proceeds in the usual fashion [LL084]. Elements of $I\!D$ are interpreted as constants; elements of C and P as states of the corresponding categories and properties, i.e., as sets and functions, respectively. More complex expressions are decomposed into simpler ones, i.e. the interpretation proceeds from the outside inwards. The interpretation of the symbols and quantors of the language yields the corresponding mathematical operations.

The data language L_F does not recognize any structure in the elements of $I\!D$. All constants are regarded as indivisible, as not having any components.

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 "?", for defining queries, the operator symbols "↑" and "↓" which denote insertion and deletion, respectively, and a connective symbol ";" to indicate composition of two updates. We first give the syntax and discuss the interpretation afterwords.

Definition 3. 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 ?, ↑, ↓ and ;. The data language $L_D = L_C \cup L_Q \cup L_U$ where the latter three sublanguages are defined in the following :

(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 3 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 intuitively clear [AER89]. The interpretation of a query is also straightforward: it is a subset of a Cartesian product. If a query contains only one variable the interpretation I in a given state s of the query has the same result as that of a set-term:

$$I_s(\ ?[\ X : c \mid q\ ]) = I_s(\{\ X : c \mid q\ \}).$$

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.

Objects can be inserted into a category subject only to the restriction that their representation is an element of the domain of the category. Removal of an object from a category is only possible when the object is not in the range of an incoming property. When the object may be removed, the corresponding ordered pair is deleted from all outgoing properties (if present). Inserting an ordered pair (x,y) into a property p may require the insertion of x into the domain category of p or of y into the range category of p, in order to stay inside the free state space. When the state of the property p already contains a pair (x, z), then this pair will be replaced by the pair (x,y). Deleting a pair from a property does not pose any special problems.

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 imposing constraints.

Definition 4. 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 $c \in C$, $a_1, \ldots, a_n \in \mathbb{D}$, $s \in S^f$ and $k \in \{\ 1, \ldots, n\text{-}1\ \}$, 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 symbols for categories and properties. We start from the scheme and replace every node with a rectangle. It 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. All lines will be labeled with the corresponding property name. The result of this step is a diagram such as Fig.1.

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.

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.

One-to-one Constraint

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

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

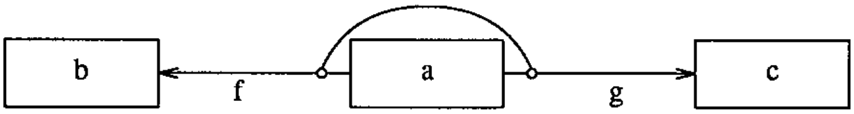
A property p satisfying a one_to_one constraint appears in the diagram with a cross bar on the property line.

Key Constraint

Consider a set A of properties with the same domain category a : $A \subseteq D^{-1}a$. For instance, let A = {f, g} with $f \in a \rightarrow b$ and $g \in a \rightarrow c$. We call these properties key properties of a when the following key constraint is satisfied :

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

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. The example above is expressed in a diagram as :

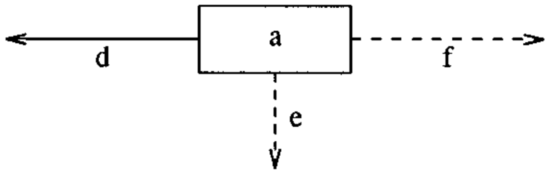


Please note, that every element in a category has a unique identification. An important difference between this intrinsic identification and identification through a set of key properties is that the intrinsic identification has to be present, otherwise the object does not exist. Key properties only 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.

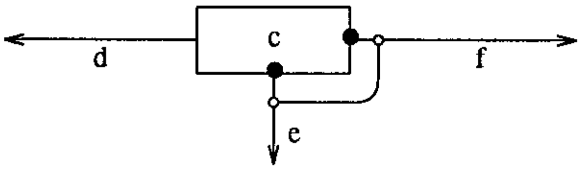
When the set A of properties is a singleton, we recover, as a special case of the key constraint, the one-to-one constraint!

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. 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 categories without outgoing properties.

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 $CB \cap CD = \emptyset$.

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 5. 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 $c \in dom(G) : G(c) \subseteq D^{-1}c$.

▯

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 6. Derivation relation

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

- $T \subseteq CD \times CD$
- $(x; y) \in T \longleftrightarrow G(x) \cap R^{-1} y \neq \emptyset$

▯

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

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

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. 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]). The similarities with this model are discussed at the end of this section. The first type is defined on the basis of scheme properties :

Definition 7. Attributes

Let E = < CB, CD, P, D, R, B, G > be an extended database scheme. A category $c \in C$ is called an attribute category if and only if $c \notin rng(D)$. []

Hence attribute categories correspond in the graphical representation of the database scheme to 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. Note, that attribute categories can already be identified at the level of a database scheme F = < C, P, D, R, V >.

The other two types of categories are defined on the basis of their representations :

Definition 8. 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
- $c \notin dom(G) \vee \forall f \in G(c) : R(f)$ 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.

[]

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.

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:

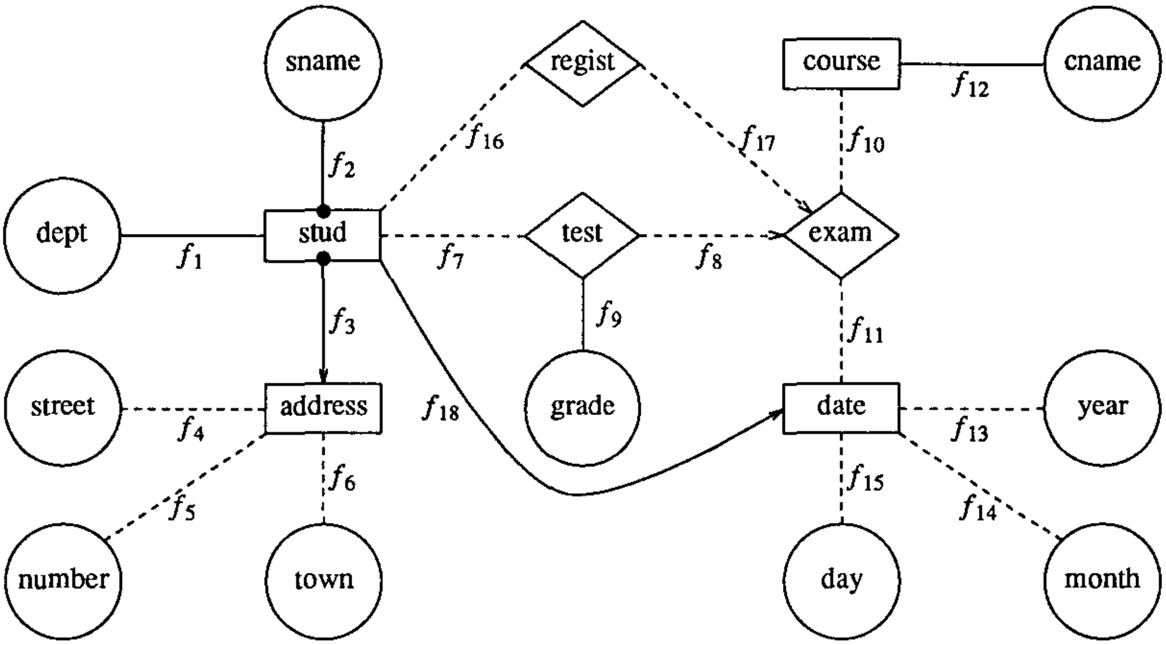


Fig.2 : Extended Scheme for a University Database

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", "regist", "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 regist has f_{16} and f_{17} as primary key functions, therefore:

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

  V(regist) = $\Pi(\{(f_{16};$ V(stud)), $(f_{17};$ V(exam))})

  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:

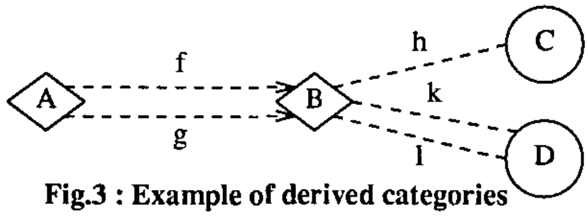


Fig.3 : Example of derived categories

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 $p \in P \backslash \cup rng(G)$ is interpreted as before : it has only a local effect. The insertion operation on a property $p \in P \backslash (R^{-1}CD \cup D^{-1}CD)$ has the same interpretation as before, but the semantics of an insertion operation on a property $p \in (R^{-1}CD \cup D^{-1}CD) \backslash \cup rng(G)$ is changed in an obvious way, when it entails an insertion into a derived category.

6. Comparison with other data models

Now that we have introduced the concepts of entity, relationship and attribute, it is easy to see that our framework can handle all concepts introduced by Chen in [CHE76]. Chen builds his data modeling framework starting from entities (the "things" that exist, concrete or abstract, in the object system) and relationships, i.e. associations among entities. Entities are classified into entity sets and relationships into relationship sets. The content of an entity set (or a relationship set) may change in the course of time by adding or removing a particular entity. The entity set therefore is a state dependent concept. The relation between entity and relationship sets is given by roles, which are named functions from a relationship set to an entity set. A relationship set is regarded as a mathematical relation, i.e. a subset of the Cartesian product of the n (not necessarily all different) entity sets involved. Alternately, it can be regarded as a subset of the generalized product, based on a set function which has as domain the set of roles, that have the relationship set as their domain, and assigns to each role the corresponding entity set. A relationship therefore can take the form of an ordered n-tuple (row) of entities, or of a set of n role-entity pairs, that is a function over the set of roles. Information about entities and relationships is recorded by means of values, which are classified into value sets. The relations between the value sets and the entity and relationship sets are given by attributes, which are named functions from an entity or relationship set to a value set or a Cartesian product of value sets. The state of a category in our framework corresponds to a (relationship, entity or value) set in the E-R-model, the state of a property corresponds to an attribute or a role. Furthermore, a value set in the E-R-model corresponds to an attribute category in our framework, and an attribute in the E-R-model corresponds to a property with an attribute category for its range.

At the next level of modeling the representations for the entities, relationships and values are chosen. Values are represented directly. In our framework this amounts to specifying the domain V(c) for an attribute category c. Entity sets in the E-R-model are identified by a set of attributes, called the primary key. Every entity is represented by a set of attribute-value pairs. This kind of representation is equivalent to that of the objects in an entity category with a derived domain in our model. The E-R-model does not discern entities which can be represented directly. In our framework we can give an entity category e with a basic domain V(e) a derived domain by removing e from CB and including e in CD, introducing an extra property p and an extra attribute category a with representation V(e), such that D(p)=e and R(p)=a and {p} is a primary key of e. Relationships in the E-R-model are identified by the entities involved. The primary key of a relationship set therefore can be represented by the primary keys of the involved entity sets. The last two forms of representation are not powerful enough and Chen therefore introduces the notions of weak entity and weak relationship. A weak entity is an entity which for its identification depends on another entity. The primary key of a weak entity set consist of attributes and of the primary key of at least one other entity set. A weak relationship set is a relationship set which is identified by a set of entities, containing at least one weak entity. With the representations described above the (regular) relationship set, the weak relationship set and the weak entity sets all are special forms of the relationship category in our framework. The implications which the introduction of weak entities and relationships has with respect to updates are similar to those introduced by the concepts of derived domains. The only difference is that Chen allows updates (an in particular deletes) of entities to propagate to all dependent relationships or entities (recursively). This would be a simple extension of our model.

The model we have presented here can be classified as 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 entities and functions (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.

Our data model is a very general and flexible model. Models such as the Entity-Relationship-model and the functional data model can be regarded as special cases, satisfying their own set of constraints. 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 <u>relational data model</u> [COD70]. In fact, a relational data model is 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.

It is easy to construct an algorithm which, given a scheme, generates a relational representation for it, satisfying the constaints above, because of the following. 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:

<u>Algorithm</u>

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)

regist (stud, course, year, month, day)

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. 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 extension of the datalanguage with constructions for handling recursion. This would, together with capabilities for handling complex objects, extend the expressive power of the datalanguage to include semantical constructions such as specialisation and association.

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.

AER89 Aerts, A.T.M. and van Hee, K.M., A Concise Formal Framework for Data Modeling, Eindhoven University of Technology Computing Science Notes CSN89.

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)

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS--circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

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.

88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.

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.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude.	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J.van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.