Algebraization and Integrity Constraints for an Extended Entity-Relationship Approach

martin gogolla
Technische Universität Braunschweig
Informatik / Datenbanken
Postfach 3329
D-3300 Braunschweig
Federal Republic of Germany
e-mail: gogolla@infbs.uucp or gogolla@dbsinf6.bitnet

Abstract:
An extended entity-relationship model concentrating nearly all concepts of known “semantic” data models and especially allowing arbitrary user defined data types is introduced. The semantics of the model is described purely in algebraic terms mainly based on the notions of signature, algebra and extension. On this basis a calculus making intensive use of abstract data types is defined and employed for the formulation of typical integrity constraints like functional restrictions and key specifications.

Keywords:
Theory of data bases, data model, entity-relationship model, formal semantics, calculus, abstract data type, aggregate function, relational completeness, integrity constraint.

1. Introduction

Among the different steps for the design of a database the conceptual design plays a mayor role [TF 82, Ce 83]. Here all requirements of later database users are collected and described in a formal way. Many authors (among them [Ch 76]) agree that the Entity-Relationship model is the most adequate data model to be used in this phase of database design. Quite a number of ER languages and ER algebras [PS 85] (where the notion algebra is used analogously to relational algebra) have been proposed, but up to now nearly no work has been done in order to define an ER calculus (again, analogously to relational calculi). It is also rather an exception [Su 87, Bii 87] in the area of databases that languages have a formal semantics, more common is the definition by examples. Special interest in the formal description of database (and especially ER) languages has to be paid to aggregate functions like this has been done for the relational approach [KI 82, ÖÖM 87].

On the other hand, some papers have tried to combine ideas developed in the field of algebraic specification (see for instance [EM 85]) with relational database design [EKW 78, DMW 82]. Also the modal systems of algebras as proposed in [GMS 83, KMS 85] supported the conceptual modelling by algebraic tools. Algebraic techniques were also successfully employed for the description of certain standard "universes" in connection with key specifications [EDG 86, SSE 87].
This paper tries to bridge the gap between the entity-relationship and the algebraic specification communities. We propose an extended ER model having an algebraic semantics and use (a polished version of) our extended ER calculus [HG 88] to formulate typical integrity constraints (to be used in the conceptual design). Let us finally remark that our calculus is not restricted to the ER model, it can be applied to other approaches [Sh 81, JS 82, SS 86] as well.

The paper is organized as follows: Chapter 2 gives an informal introduction into the data model and the calculus by means of an example. Chapter 3 formally defines the model and points out how to formulate it in algebraic terms. In chapter 4 the calculus is defined and it is applied in chapter 5 to formulate integrity constraints.

2. The Basic Idea

Before we explain the formal details of our approach we point out the basic ideas by means of an example. We consider a simple geo-scientific database where information about towns, countries and rivers has to be stored. First, we have entity types which are described by attributes returning values of given data types (possibly set-, bag- or list-valued), e.g., a TOWN has a name, a population and a geometry.

Secondly, relationships can exist between entity types, e.g., RIVERS flow through COUNTRIES, TOWNS lie at RIVERS and lie in COUNTRIES. These relationships can also have attributes.

Thirdly, entity types can have other entity types as components, e.g., a TOWN has as component the set of its DISTRICTS and each district has as component the set of STREETS lying in the district.

Last, but not least, entity types can be constructed from other entity types, e.g., an entity of type WATERS is constructed from an entity of type RIVER, SEA or LAKE. RIVER, SEA and LAKE are the input types of the construction, WATERS the output type.
Due to space limitations only a short sketch of our model can be given. Full motivation of all concepts can be found in \[HG 88\]. For an entity-relationship schema (as introduced above) we shall define a signature (in the sense of abstract data type theory) and a database state for such a schema will then be an algebra for this signature (with carrier sets, functions and relations).

We will also define a calculus for such entity-relationship schemas which can be employed to express queries and integrity constraints. The calculus especially distinguishes between sets and bags (multisets). Therefore it is well suited to formulate aggregation properties not expressible for instance in the "classical" relational tuple or domain calculus \[Ma 83\]. For example the query "Give me for each country its name and the average of the population of its towns" is formulated as

\[
\{ \text{name(c)} , \text{AVG} \{ \text{population(t)} | \text{t : TOWN} \land \text{lies-in(t,c)} \} | \text{c : COUNTRY} \}
\]

Terms of the form \{ \ldots \} are bag-valued (retaining duplicates). Please notice, this is essential for the calculation of the average in the subquery, if two towns have the same population. It is also possible to restrict variables to the finite set of all stored values: Assume 'government' is an attribute for COUNTRY of sort \text{string} (for instance "socialistic", "democratic", etc.). Then the query "Give me every (stored) form of government and for every (stored) form of government the sum of country populations having this form" will be expressed as

\[
\{ \text{g} , \text{SUM} \{ \text{population(c)} | \text{c : COUNTRY} \land \text{government(c)=g} \} | \text{g : BTS} \{ \text{government(c)} | \text{c : COUNTRY} \} \}
\]

The standard function BTS converts a Bag To a Set. The calculus is also employed to formulate integrity constraints. For example, relationships can be required to be functional:

\[
( \forall \text{fi} , \text{fi'} : \text{flows-into} ) \quad \text{fi}.\text{RIVER} = \text{fi'}.\text{RIVER} \Rightarrow \text{fi}.\text{WATERS} = \text{fi'}.\text{WATERS}
\]

This formula means that \text{flows-into} is functional from \text{RIVER} to \text{WATERS}, i.e., a river flows into (at most) one water. Another application is the specification of key attributes, key components and key relations:

\[
( \forall \text{r} , \text{r'} : \text{RIVER} ) \\
\text{r} \neq \text{r'} \Rightarrow \\
\text{name(r)} \neq \text{name(r')} \lor ( \exists \text{c : COUNTRY} ) \text{flows-through(r,c)} \text{ xor } \text{flows-through(r',c)}
\]

This formula, where \text{xor} stands for the exclusive or, expresses that a river is identified by its name and the countries through which it flows: \{\text{name,flows-through}\} is a key for \text{RIVER}. In other words, two different rivers must have different names or (if their names coincide) there must be a country such that one river flows through this country and the other one not: the rivers have to be observably inequivalent with respect to "name" or "flows-through". Let us finally mention the possibility of cardinality constraints:

\[
( \forall \text{d : DISTRICT} ) \\
50 \leq \text{CNT(streets(d))} \land \text{CNT(streets(d))} \leq 100
\]

The above line (\text{CNT} stands for \text{COUNT}) says that districts have at least 50 and at most 100 streets.

3. Algebraization of the Extended Entity-Relationship Model

3.1 Axiomatic conventions: Let \text{iset} denote the class of sets, \text{ifiset} the class of finite sets, \text{ifun} the class of total functions and \text{irel} the class of relations. There are the obvious
inclusions $|\text{FIN}| \subseteq |\text{SET}|$ and $|\text{FUN}| \subseteq |\text{REL}| \subseteq |\text{SET}|$. Assume sets $S, S_1, \ldots, S_n \in |\text{SET}|$ are given. Then $F(S)$ denotes the restriction of the powerset $P(S)$ of $S$ to finite sets, $S^*$ the set of finite lists over $S$, $S^+$ the set of finite non-empty lists over $S$, and $S_1 \times \ldots \times S_n$ the cartesian product of the sets $S_1, \ldots, S_n$. The set of finite multisets or bags over $S$ is given by $B(S)$. A bag can be considered as a (finite) set $S$ together with a counting function $\text{occur} : S \to \mathbb{N}$, giving for each element the number of occurrences in the bag.

Finite sets are written as $\{c_1, \ldots, c_n\}$, lists as $<c_1, \ldots, c_n>$, elements of the cartesian product as $(c_1, \ldots, c_n)$, and bags as $\{(c_1, \ldots, c_n)\}$. For a set $\{c_1, \ldots, c_n\}$, $i \neq j$ implies $c_i \neq c_j$. But this is not necessarily true for bags: If we have a bag $\{(c_1, \ldots, c_n)\}$ with $\text{occur}(c)=k$, this implies that there are $k$ distinct indices $i_1, \ldots, i_k \in \{1\ldots n\}$ with $c_{i_j} = c$ for $j \leq k$.

3.2 Definition: (data signature)

The syntax of a data signature $DS$ is given by
- the sets DATA, OPNS, PRED $\in |\text{FIN}|$,
- a function $\text{source} : \text{OPNS} \to \text{DATA}^*$,
- a function $\text{destination} : \text{OPNS} \to \text{DATA}$, and
- a function $\text{arguments} : \text{PRED} \to \text{DATA}^+$.

If $o \in \text{OPNS}$, $\text{source}(o) = <d_1, \ldots, d_n>$, and $\text{destination}(o) = d$, this is notated as $o : d_1 \times \ldots \times d_n \rightarrow d$. If $\pi \in \text{PRED}$ with $\text{arguments}(\pi) = <d_1, \ldots, d_n>$, this is notated as $\pi : d_1 \times \ldots \times d_n$.

The semantics of a data signature $DS$ is given by
- a function $\mu[\text{DATA}] : \text{DATA} \rightarrow |\text{FIN}|$ and $\bot \in \mu[\text{DATA}](d)$ for every $d \in \text{DATA}$,
- a function $\mu[\text{OPNS}] : \text{OPNS} \rightarrow |\text{FUN}|$ and $o : d_1 \times \ldots \times d_n \rightarrow d$ implies $\mu[\text{OPNS}](o) : \mu[\text{DATA}](d_1) \times \ldots \times \mu[\text{DATA}](d_n) \rightarrow \mu[\text{DATA}](d)$ for every $o \in \text{OPNS}$, and
- a function $\mu[\text{PRED}] : \text{PRED} \rightarrow |\text{REL}|$ and $\pi : d_1 \times \ldots \times d_n$ implies $\mu[\text{PRED}](\pi) \subseteq \mu[\text{DATA}](d_1) \times \ldots \times \mu[\text{DATA}](d_n)$ for every $\pi \in \text{PRED}$.

The set $\text{OPNS}^2$ denotes all operators $o, o \in \text{OPNS}$, having source$(o) = <d,d>$ and destination$(o) = d$ for some $d \in \text{DATA}$. Additionally the corresponding function $\mu[\text{OPNS}](o)$ has to be commutative and associative.

3.3 Remarks: Throughout the paper all sets mentioned in definitions have to be disjoint, except when common elements are explicitly allowed. Thus, e.g., DATA, OPNS, and PRED are disjoint as well as the interpretations of all data sorts with the exception of $\bot$, i.e., $\mu[\text{DATA}](d_1) \cap \mu[\text{DATA}](d_2) = \{\bot\}$, if $d_1 \neq d_2$. Furthermore, throughout the paper the greek letter $\mu$ stands for meaning (and hopefully not for mysterious).

We have required every data sort $d$ to contain the value $\bot \in \mu[\text{DATA}](d)$, because it is useful to have an 'undefined' value as result for incorrect applications of operations. Thus, we obtain for incorrect applications of operations this special value, for example $\mu[\text{OPNS}](/)(c,0) = \bot$. In most cases it is useful to define the propagation of $\bot$ in the following way: An operation $o \in \text{OPNS}$ with $o : d_1 \times \ldots \times d_n \rightarrow d$ evaluates $\mu[\text{OPNS}](o)(c_1, \ldots, c_n)$ to $\bot$, if there is a $c_i$ with $c_i = \bot$. For predicates $\pi \in \text{PRED}$ with $\pi : d_1 \times \ldots \times d_n$ $(c_1, \ldots, c_n) \in \mu[\text{PRED}](\pi)$ does not hold, if there is a $c_i$ with $c_i = \bot$.

For the rest of the paper we assume one fixed data signature and algebra including (among others) the sorts int, rat (for integer and rational numbers) and string together with adequate operations and predicates to be given.
3.4 Example: Our example presented in chapter 2 uses the data sorts int, rat, string and point. As will be seen later in the context of aggregations like SUM or AVG, the addition + on integers and on rationals has to be in the set OPNS^{2,1}. We also need the division / on rationals.

3.5 Fact: The syntax of the data signature directly corresponds to a signature DS = (DATA, OPNS, PRED) in the sense of abstract data type theory. The semantics of a data signature is equivalent to an algebra \( \mu[DS] = (\mu[DATA], \mu[OPNS], \mu[PRED]) \) with carrier sets, functions and relations.

3.6 Definition: (sort expressions)

Let a data signature DS and a set \( S \subseteq S \) together with a (semantic) function \( \mu(S) : S \rightarrow |SET| \) (such that \( \mu(S)(d) = \mu[DATA](d) \) for \( d \in DATA \) and \( 1 \in \mu(S)(s) \) for every \( s \in S \)) be given. The syntax of the sort expressions over \( S \) is given by the set \( \text{SORT-EXPR}(S) \) determined by the following rules.

(i) If \( s \in S \), then \( s \in \text{SORT-EXPR}(S) \).

(ii) If \( s \in \text{SORT-EXPR}(S) \), then \( \text{set}(s) \in \text{SORT-EXPR}(S) \).

(iii) If \( s \in \text{SORT-EXPR}(S) \), then \( \text{list}(s) \in \text{SORT-EXPR}(S) \).

(iv) If \( s \in \text{SORT-EXPR}(S) \), then \( \text{bag}(s) \in \text{SORT-EXPR}(S) \).

(v) If \( s_1, \ldots, s_n \in \text{SORT-EXPR}(S) \), then \( \text{prod}(s_1, \ldots, s_n) \in \text{SORT-EXPR}(S) \).

The semantics of the sort expressions is a function \( \mu[\text{SORT-EXPR}(S)] : \text{SORT-EXPR}(S) \rightarrow |SET| \) determined by the following rules.

(i) \( \mu[\text{SORT-EXPR}(S)](s) := \mu(S)(s) \)

(ii) \( \mu[\text{SORT-EXPR}(S)](\text{set}(s)) := F(\mu[\text{SORT-EXPR}(S)](s)) \cup \{1\} \)

(iii) \( \mu[\text{SORT-EXPR}(S)](\text{list}(s)) := (\mu[\text{SORT-EXPR}(S)](s))^* \cup \{1\} \)

(iv) \( \mu[\text{SORT-EXPR}(S)](\text{bag}(s)) := B(\mu[\text{SORT-EXPR}(S)](s)) \cup \{1\} \)

(v) \( \mu[\text{SORT-EXPR}(S)](\text{prod}(s_1, \ldots, s_n)) := (\mu[\text{SORT-EXPR}(S)](s_1) \times \ldots \times \mu[\text{SORT-EXPR}(S)](s_n)) \cup \{1\} \)

\( \mu[\text{SORT-EXPR}(S)] \) is an abbreviation for \( \bigcup_{s \in \text{SORT-EXPR}(S)} \mu[\text{SORT-EXPR}(S)](s) \), i.e., the set of all instances belonging to the sort expression over the set \( S \).

3.7 Example: In our example \( S \) is equal to the set \{int, rat, point, string, TOWN, RIVER, COUNTRY, SEA, LAKE, WATERS, DISTRICT, STREET\}. The diagrams use three multi-valued sort expression, namely list(point), set(DISTRICT) and set(STREET).

3.8 Fact: The set \( S \) together with the semantic function \( \mu(S) \) induces a signature \( DS \cup S = (S, OPNS, PRED) \) and an algebra \( \mu[DS \cup S] = (\mu[S], \mu[OPNS], \mu[PRED]) \), which is a conservative and complete extension of \( \mu[DS] \) (due to the fact that \( \mu[DS](d) = \mu[S](d) \) for \( d \in DATA \)).

3.9 Remark: The same semantics of sort expressions can also be specified algebraically, if we view a sort expression as a new sort and introduce generating operations in the following way.

\( \text{EMPTY}_{\text{set}(a)} : \rightarrow \text{set}(s) \)

\( \text{ADD}_{\text{set}(a)} : \text{set}(s) \times s \rightarrow \text{set}(s) \)
EMPTY \textit{bag(s)} : \rightarrow \textit{bag(s)}

\text{ADD}_{\textit{bag(s)}} : \textit{bag(s)} \times s \rightarrow \textit{bag(s)}

EMPTY \textit{list(s)} : \rightarrow \textit{list(s)}

\text{ADD}_{\textit{list(s)}} : \textit{list(s)} \times s \rightarrow \textit{list(s)}

\text{MAKE}_{\text{prod}(s_1, ..., s_n)} : s_1 \times ... \times s_n \rightarrow \text{prod}(s_1, ..., s_n)

Additionally, error constants representing \(\perp\) could be added. For sets, two equations must be valid, whereas the interpretation of bags is restricted by only one equation. Lists and products are generated "freely" (without restricting equations).

\text{ADD}_{\textit{set}(s)}(\text{ADD}_{\textit{set}(s)}(S,X),Y) = \text{ADD}_{\textit{set}(s)}(\text{ADD}_{\textit{set}(s)}(S,Y),X)

\text{ADD}_{\textit{set}(s)}(\text{ADD}_{\textit{set}(s)}(S,X),X) = \text{ADD}_{\textit{set}(s)}(S,X)

\text{ADD}_{\textit{bag}(s)}(\text{ADD}_{\textit{bag}(s)}(S,X),Y) = \text{ADD}_{\textit{bag}(s)}(\text{ADD}_{\textit{bag}(s)}(S,Y),X)

Because the second set equation is not valid for bags, bags and sets fulfill the following rules (with respect to the \{\{...\}\} and \{...\}-notation).

\{\{c_1\}\} \cup \{\{c_2\}\} = \{\{c_1, c_2\}\} = \{\{c_2, c_1\}\} \text{ if } c_1 \neq c_2

\{\{c\}\} \cup \{\{c\}\} = \{\{c, c\}\}

\{\{c_1\}\} \cup \{\{c_2\}\} = \{\{c_1, c_2\}\} = \{\{c_2, c_1\}\} \text{ if } c_1 \neq c_2

\{\{c\}\} \cup \{\{c\}\} = \{\{c, c\}\}

\section{3.10 Definition: (operations and predicates induced by the sort expressions)}
Let the sort expressions as defined above be given. The \textbf{syntax of the operations and predicates induced by the sort expressions} is given by the following sets and functions (s, s_1, ..., s_n refer to arbitrary sort expressions and o to an element of \textit{OPNS}^2 with o : d x d \rightarrow d):

\begin{tabular}{llll}
\text{CNT}_{\textit{set}(s)} & \text{: set(s)} & \rightarrow & \text{int} \\
\text{IND}_{\textit{set}(s)} & \text{: set(s)} & \rightarrow & \text{set(int)} \\
\text{APL}_{o, \textit{set}(d)} & \text{: set(d)} & \rightarrow & d \\
\text{IN}_{\textit{set}(s)} & \text{: prod(set(s),s)} & \rightarrow & \text{set(int)} \\
\hline 
\text{CNT}_{\textit{list}(s)} & \text{: list(s)} & \rightarrow & \text{int} \\
\text{IND}_{\textit{list}(s)} & \text{: list(s)} & \rightarrow & \text{set(int)} \\
\text{LTB}_{\textit{list}(s)} & \text{: list(s)} & \rightarrow & \text{bag(s)} \\
\text{SEL}_{\textit{list}(s)} & \text{: prod(list(s),int)} & \rightarrow & s \\
\text{POS}_{\textit{list}(s)} & \text{: prod(list(s),s)} & \rightarrow & \text{set(int)} \\
\text{APL}_{o, \textit{list}(d)} & \text{: list(d)} & \rightarrow & d \\
\text{IN}_{\textit{list}(s)} & \text{: prod(list(s),s)} & \rightarrow & \text{set(int)} \\
\hline 
\text{CNT}_{\textit{bag}(s)} & \text{: bag(s)} & \rightarrow & \text{int} \\
\text{IND}_{\textit{bag}(s)} & \text{: bag(s)} & \rightarrow & \text{set(int)} \\
\text{BTS}_{\textit{bag}(s)} & \text{: bag(s)} & \rightarrow & \text{set(s)} \\
\text{OCC}_{\textit{bag}(s)} & \text{: prod(bag(s),s)} & \rightarrow & \text{int} \\
\text{APL}_{o, \textit{bag}(d)} & \text{: bag(d)} & \rightarrow & d \\
\text{IN}_{\textit{bag}(s)} & \text{: prod(bag(s),s)} & \rightarrow & \text{set(int)} \\
\text{PRJ}_{\text{prod}(s_1, ..., s_n)} & \text{: prod(s_1, ..., s_n)} & \rightarrow & s_i \text{ (i.d.,n)}
\end{tabular}

The \textbf{semantics of the operations induced by the sort expressions} is a function \(\mu[\text{OPNS}(S)] : \text{OPNS}(S) \rightarrow \text{IFUN}\) and a function \(\mu[\text{PRED}(S)] : \text{PRED}(S) \rightarrow \text{REL}\) determined by the following lines. Domain and codomain of semantic functions and predicates are not given explicitly,
but are determined by the following rules: If $o : s_1 \rightarrow s_2$, then $\mu[\text{OPNS}(S)](o) : \mu[\text{SORT-EXPR}(S)](s_1) \rightarrow \mu[\text{SORT-EXPR}(S)](s_2)$, and if $\pi : s$, then $\mu[\text{PRED}(S)](\pi) \subset \mu[\text{SORT-EXPR}(S)](s)$. Furthermore we abbreviate $\mu[\text{OPNS}(S)](o)$ by $\mu(o)$ and $\mu[\text{PRED}(S)](\pi)$ by $\mu(\pi)$. All these functions preserve the undefined value $\perp$ ($\mu(\perp) = \perp$ and $\mu(o)(c_1,c_2) = \perp$, if $c_1 = \perp$ or $c_2 = \perp$) and the predicates do not hold for $\perp$ (not $\perp \in \mu(\pi)$).

$\mu[\text{CNT}(S)] : \{c_1,...,c_n\} \mapsto n$

$\mu[\text{IND}(S)] : \{c_1,...,c_n\} \mapsto \{1,...,n\}$

$\mu[\text{APL}(S)] : \{c_1,...,c_n\} \mapsto \begin{cases} \perp & \text{if } n=0 \\ c_1 & \text{if } n=1 \\ \{\mu(o)(c_1,\mu[\text{APL}(S)](\{c_2,...,c_n\})) \} & \text{if } n>2 \end{cases}$

$\mu[\text{IN}(S)] : \{\{c_1,...,c_n\}, c\} \mapsto \{c_1,...,c_n\}$

$\mu[\text{LTB}(S)] : \{c_1,...,c_n\} \mapsto \{\{c_1,...,c_n\}\}$

$\mu[\text{SEL}(S)] : \{\{c_1,...,c_n\}, i\} \mapsto \begin{cases} c_i & \text{if } 1 \leq i \leq n \\ \perp & \text{otherwise} \end{cases}$

$\mu[\text{POS}(S)] : \{\{c_1,...,c_n\}, c\} \mapsto f[\text{list}(S)](\{c_1,...,c_n\}, c, \{\})$

The function $f[\text{list}(S)] : \text{list}(S) \times \text{set}(\text{int}) \rightarrow \text{set}(\text{int})$ is not an operation induced by the sort expressions. It is only part of the definition of $\text{POS}$:

$f[\text{list}(S)] : \{\{c_1,...,c_n\}, c, r\} \mapsto \begin{cases} r & \text{if } n=0 \\ f[\text{list}(S)](\{c_1,...,c_{n-1}\}, c, r) & \text{if } n=1 \text{ and } c \neq c \\ f[\text{list}(S)](\{c_1,...,c_{n-1}\}, c, r \cup \{n\}) & \text{if } n=1 \text{ and } c = c \\ \{\} & \text{if } n=0 \end{cases}$

$\mu[\text{BTS}(S)] : \{\{c_1,...,c_n\}\} \mapsto \{\{c_1\} \cup \mu[\text{BTS}(S)](\{\{c_2,...,c_n\}\}) \text{ if } n>1$

$\mu[\text{OCC}(S)] : \{\{c_1,...,c_n\}, c\} \mapsto \begin{cases} 0 & \text{if } n=0 \\ \mu[\text{OCC}(S)](\{\{c_2,...,c_n\}\}, c) & \text{if } n=1 \text{ and } c \neq c \\ \mu[\text{OCC}(S)](\{\{c_2,...,c_n\}\}, c+1) & \text{if } n=1 \text{ and } c = c \\ \{\} & \text{if } n=0 \end{cases}$

$\mu[\text{PROJ}(S_1,...,S_n)] : \{c_1,...,c_n\} \mapsto c_i$

$\mu[\text{APL}(S)]$ and $\mu[\text{APL}(S)]$ are defined just as $\mu[\text{APL}(S)]$ (analogously for the other functions not mentioned explicitly).

3.11 Remarks: $\{...\} \cup \{...\}$ refers to the union of sets (this guarantees the elimination of duplicates, while $\{\{...\}\} \cup \{...\}$ is the union of bags respecting duplicates). For all above functions equational specifications can be given. We use the following conventions and abbreviations for frequently used operations. If no ambiguities occur, we drop the list-, set-, bag- or prod-indices of the operation symbols and leave out parenthesis in accordance with the usual rules. $\text{SUM}$ refers to $\text{APL}_+$ dependent on the context (the idea of applying a binary operator to a list of values has also been proposed in [Bi 87]). $\text{MAX}$ means $\text{APL}_+ \text{ max}$ with $\text{max}(x,y) = \begin{cases} x & \text{if } x \geq y \\ y & \text{else} \end{cases}$. $\text{MIN}$ is defined analogously. $\text{AVG}(\text{set})$ refers to $\text{APL}_+ \text{ set}(\text{set}) \text{ / } \text{CNT}(\text{set})$, analogously for lists and bags. With this convention we have of course $\text{AVG}(\perp) = \perp$. Instead of $\text{PROJ}_i(x)$ or $\text{PRJ}(\{x_1,...,x_n\})$ we also use the more suggestive $x.i$ or $(x_1,...,x_n).i$, where $i$ is a constant between 1 and n. $\text{LTS}$ stands for $\text{BTS} \circ \text{LTB}$. 
3.12 **Example:** In the first query of the example presented in chapter 2 AVG \{ ... \} stands for APL_{bag(int)} \{ ... \} / CNT_{bag(int)} \{ ... \}. In the second query SUM refers to APL_{bag(int)} and BTS was short for BTS_{bag(string)}.

3.13 **Fact:** The syntax of the sort expressions introduces a signature

\[ DS \vee SORT-EXPR(S) = (SORT-EXPR(S), \text{OPNS} \vee \text{OPNS}(S), \text{PRED} \vee \text{PRED}(S)) \]

and the semantics correspond to an algebra

\[ \mu[DS \vee SORT-EXPR(S)] = \mu[SORT-EXPR(S)], \mu[\text{OPNS} \vee \text{OPNS}(S)], \mu[\text{PRED} \vee \text{PRED}(S)]. \]

Again, \( \mu[DS \vee SORT-EXPR(S)] \) is a conservative and complete extension of \( \mu[DS \vee S] \).

Of course, there is an infinite number of sorts and operations in this algebra.

3.14 **Definition:** (extended entity-relationship schema)

Let a data signature DS be given. The **syntax of an extended entity-relationship schema EE(R) (DS)** over DS is given by

- the sets ENTITY, RELATION, ATTRIBUTE, COMPONENT, CONSTRUCTION \( \in \mathcal{IFSET} \) and
- the functions participants, asource, adestination, csource, cdestination, input, output such that

<table>
<thead>
<tr>
<th>Function</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>participants</td>
<td>RELATION \rightarrow ENTITY +</td>
<td></td>
</tr>
<tr>
<td>asource</td>
<td>ATTRIBUTE \rightarrow ENTITY \cup RELATION</td>
<td></td>
</tr>
<tr>
<td>adestination</td>
<td>ATTRIBUTE \rightarrow { d, set(d), list(d), bag(d) \ \mid d \in \text{DATA} }</td>
<td></td>
</tr>
<tr>
<td>csource</td>
<td>COMPONENT \rightarrow ENTITY</td>
<td></td>
</tr>
<tr>
<td>cdestination</td>
<td>COMPONENT \rightarrow { e, set(e), list(e), bag(e) \ \mid e \in \text{ENTITY} }</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>CONSTRUCTION \rightarrow F(ENTITY)</td>
<td>and</td>
</tr>
<tr>
<td>output</td>
<td>CONSTRUCTION \rightarrow F(ENTITY)</td>
<td></td>
</tr>
</tbody>
</table>

If \( r \in \text{RELATION} \) with \( \text{participants}(r) = \langle e_1, ..., e_n \rangle \), this is notated as \( r(e_1, ..., e_n) \). If \( a \in \text{ATTRIBUTE} \) with \( \text{asource}(a) = e \) or \( \text{asource}(a) = r \) and \( \text{adestination}(a) = d \), this is notated as \( a : e \rightarrow d \) or \( a : r \rightarrow d \), respectively. If \( c \in \text{COMPONENT} \) with \( \text{csource}(c) = e \) and \( \text{cdestination}(c) = e' \), this is notated as \( c : e \rightarrow e' \). If \( c \in \text{CONSTRUCTION} \) with \( \text{input}(c) = \{ i_1, ..., i_n \} \) and \( \text{output}(c) = \{ o_1, ..., o_m \} \), this is notated as \( c(i_1, ..., i_n ; o_1, ..., o_m) \).

For two distinct constructions \( c_1, c_2 \in \text{CONSTRUCTION} \) the following conditions must hold:

(i) \( \text{output}(c_1) \cap \text{output}(c_2) = \emptyset \)

(ii) It is not allowed that \( \text{connection}^+(e, e) \) holds for some \( e \in \text{ENTITY} \), where \( \text{connection}^+ \) is the transitive closure of the relation \( \text{connection} \) defined by: if \( e_{\text{in}} \in \text{input}(c) \) and \( e_{\text{out}} \in \text{output}(c) \) for some \( c \in \text{CONSTRUCTION} \), then \( \text{connection}(e_{\text{in}}, e_{\text{out}}) \) holds.

**The semantics of an extended entity-relationship schema EE(RDS) (DS)** is given by

- a function \( \mu[\text{ENTITY}] : \text{ENTITY} \rightarrow \mathcal{IFSET} \) such that \( \perp \in \mu[\text{ENTITY}](e) \) for \( e \in \text{ENTITY} \),

- a function \( \mu[\text{RELATION}] : \text{RELATION} \rightarrow \mathcal{IFSET} \) such that \( r(e_1, ..., e_n) \) implies \( \mu[\text{RELATION}](r) \subseteq \{ \mu[\text{ENTITY}](e_1) \times ... \times \mu[\text{ENTITY}](e_n) \} \cup \{ \perp \} \), and \( \perp \in \mu[\text{RELATION}](r) \) for \( r \in \text{RELATION} \),

- a function \( \mu[\text{ATTRIBUTE}] : \text{ATTRIBUTE} \rightarrow \mathcal{IF} \) such that

\[ a : e \rightarrow d \text{ implies } \mu[\text{ATTRIBUTE}](a) : \mu[\text{ENTITY}](e) \rightarrow \mu[\text{SORT-EXPR}](\text{DATA}))(d), \]

\[ a : r \rightarrow d \text{ implies } \mu[\text{ATTRIBUTE}](a) : \mu[\text{RELATION}](r) \rightarrow \mu[\text{SORT-EXPR}](\text{DATA}))(d). \]

- a function \( \mu[\text{COMPONENT}] : \text{COMPONENT} \rightarrow \mathcal{IF} \) such that

\[ c : e \rightarrow e' \text{ implies } \mu[\text{COMPONENT}](c) : \mu[\text{ENTITY}](e) \rightarrow \mu[\text{SORT-EXPR}](\text{ENTITY}))(e'), \]

- a function \( \mu[\text{CONSTRUCTION}] : \text{CONSTRUCTION} \rightarrow \mathcal{IF} \) such that \( c(i_1, ..., i_n \circ o_1, ..., o_m) \) implies \( \mu[\text{CONSTRUCTION}](c) : \bigcup_{k=1}^n \mu[\text{ENTITY}](o_k) \rightarrow \bigcup_{k=1}^n \mu[\text{ENTITY}](i_k) \),

where each \( \mu[\text{CONSTRUCTION}](c) \) is injective.
Every function $\mu[\text{ATTRIBUTE}](a)$, $\mu[\text{COMPONENT}](c)$, and $\mu[\text{CONSTRUCTION}](c)$ has to preserve the undefined value, e.g., $\mu[\text{ATTRIBUTE}](a)(\bot) = \bot$ and so on.

3.15 Remarks: The elements of ENTITY are called entity (or object) types, the elements of RELATION are the relationship types, and the elements of ATTRIBUTE are the attributes names, all known from the ER Model defined in [Ch 76]. $\mu[\text{ENTITY}](e)$ is the set of entities belonging to the entity type $e$, $\mu[\text{RELATION}](r)$ defines which entities are related by $r$, and $\mu[\text{ATTRIBUTE}](a)$ gives the attributes of entities or relationships (related entities).

Additionally, we have a set COMPONENT of component names to model complex entity types. $\mu[\text{COMPONENT}](c)$ gives the components of entities, i.e., an entity may have as part (component) of itself another entity, a set, list, or bag of entities.

To provide modelling primitives for specialization and generalization [SS 77] we introduce the set CONSTRUCTION. Its elements are called type constructions. A type construction $c$ may be regarded as a rearrangement of entity types. Starting with non-constructed or already defined entity types in $\text{input}(c)$ the new entity types in $\text{output}(c)$ are constructed.

Although all introduced entity types are disjoint by definition, the constructed entity types may be considered as a new classification of the entities in the input types. Formally, we express this fact by the function

$$\mu[\text{CONSTRUCTION}](c) : \bigcup_{j=1}^{m} \mu[\text{ENTITY}](o_j) \rightarrow \bigcup_{k=1}^{n} \mu[\text{ENTITY}](i_k)$$

yielding for an output entity the corresponding input entity it refers to. Since this function is injective, every output entity corresponds to exactly one input entity. But an input entity need not appear in any output type at all, because the function is not required to be surjective. This semantics of CONSTRUCTION is equivalent to demanding that every output entity "is" an input entity:

$$\bigcup_{j=1}^{m} \mu[\text{ENTITY}](o_j) \subseteq \bigcup_{k=1}^{n} \mu[\text{ENTITY}](i_k)$$

The restricting conditions for constructions guarantee that we have for a constructed entity $e$ uniquely determined constructions $c_1, ..., c_n$, other constructed entities $e_1, ..., e_{n-1}$ and a non-constructed (basic) entity $e_n$, such that

$$e \xrightarrow{\mu[c_1]} e_1 \xrightarrow{\mu[c_2]} e_2 \xrightarrow{\mu[c_3]} e_3 \cdots e_{n-1} \xrightarrow{\mu[c_n]} e_n$$

In other words, for a constructed entity there is a uniquely determined construction rule.

As for DATA, we assume the element $\bot$ to be element of every entity type in ENTITY. Thus, every attribute and component is per default optional. Indeed, this can be excluded by additional integrity constraints.

3.16 Example: In our example the following identities hold: ENTITY = \{TOWN, RIVER, COUNTRY, SEA, LAKE, WATERS, DISTRICT, STREET\}, RELATION = \{lies-at, lies-in, flows-through, flows-into\}, COMPONENT = \{districts, streets\} and CONSTRUCTION = \{are\}. The corresponding function participants, etc. can be derived from the diagrams.

3.17 Fact: The algebraization of an extended entity-relationship schema will be done in two steps. First we introduce the signature

$$\text{EER-BASE} = (\text{SORT-EXPR(DATA)} \cup \text{ENTITY} \cup \text{RELATION},$$
$$\text{OPNS} \cup \text{OPNS(DATA)} \cup \text{OPNS(\text{EER-BASE})},$$
$$\text{PRED} \cup \text{PRED(DATA)} )$$

with


OPNS(ERR-BASE) = \{ mk_r : e_1 \times \ldots \times e_n \rightarrow r | r(e_1,...,e_n) \in EER(DS) \} \cup \\
\{ a : e \rightarrow d \in EER(DS), a : r \rightarrow d \in EER(DS) \} \cup \\
\{ c_{ij} : o_j \rightarrow i_k | c(i_1,...,o_n) \in EER(DS), j \leq m, k \leq n \}

Given the semantics of an EER schema, an algebra \( \mu[ERR-BASE] \) can be defined as follows. The carriers are given by \( \mu[SORT-EXPR(DATA)] \) (which is based on \( \mu[DATA] \)), \( \mu[ENTITY] \) and \( \mu[RELATION] \). The functions \( \mu[OPNS] \) and \( \mu[OPNS(DATA)] \) and the predicates \( \mu[PRED] \) and \( \mu[PRED(DATA)] \) are fixed by the algebra \( \mu[DS \cup SORT-EXPR(DATA)] \). It remains to define the functions \( \mu[OPNS(ERR-BASE)] \):

- \( \mu(mk_r) : (e_1,...,e_n) \rightarrow \chi(e_1,...,e_n) \in r \) then \( \chi(e_1,...,e_n) \) else \( \mu[OPNS(ERR-BASE)](a) \) is given by \( \mu[ATTRIBUTE](a) \).
- \( \mu(c_{ij}) : o_j \rightarrow \chi \) if \( \mu[CONSTRUCTION](c)(o_j) \in \chi \) then \( \mu[CONSTRUCTION](c)(o_j) \) else \( \mu[CONSTRUCTION](c)(o_j) \).

In the second step we define the final signature by

\[ EER = (SORT-EXPR(DATA \cup ENTITY) \cup RELATION, \]
\[ \text{OPNS} \cup \text{OPNS}(DATA \cup ENTITY) \cup \text{OPNS(ERR-BASE)} \cup \text{OPNS}(ERR), \]
\[ \text{PRED} \cup \text{PRED}(DATA \cup ENTITY) ) \] with

\[ \text{OPNS}(ERR) = \{ c : e \rightarrow e' \in EER(DS) | c \in \text{COMPONENT} \}. \]

The corresponding algebra \( \mu[ERR] \) has carriers equal to \( \mu[ERR-BASE] \) except that the additional carriers are determined by \( \mu[SORT-EXPR(DATA \cup ENTITY)] \) (which is based on \( \mu[DATA] \) of the DS level and \( \mu[ENTITY] \) of the ERR-BASE level). The additional functions in \( \text{OPNS}(ERR) \) are determined by \( \mu[COMPONENT](c) \).

Again, the considered algebras are strongly related: \( \mu[ERR] \) is a conservative and complete extension of \( \mu[ERR-BASE] \) which extends \( \mu[DS \cup SORT-EXPR(DATA)] \).

3.18 Example: In the signature EER for our example we will have (among others) the following sorts and functions.

name : TOWN \rightarrow \text{string} ;
population : TOWN \rightarrow \text{int} ;
graphy : TOWN \rightarrow \text{list(point)} ;

\( mk\text{flows-through} : \text{RIVER} \times \text{COUNTRY} \rightarrow \text{flows-through} \)
length : \text{flows-through} \rightarrow \text{rat} ;
districts : TOWN \rightarrow \text{set(DISTRICT)} ;

\( \text{areWATERS:RIVER} : \text{WATERS} \rightarrow \text{RIVER} ; \)
\( \text{areWATERS:SEA} : \text{WATERS} \rightarrow \text{SEA} \)

4. The Entity-Relationship Calculus

4.1 Notations: Before discussing the EER calculus in more detail, we introduce some abbreviations. Let an EER schema EER(DS) be given. SORT refers to the union of all data, entity, and relationship sorts : SORT := DATA \cup ENTITY \cup RELATION. Furthermore, we extend the notions of operations (OPNS) and predicates (PRED) introduced for data signatures to sort expressions and EER schemas resulting in the following notations:

\[ \text{OPNS}_{\text{EXPR}} := \{ \text{CNT, LTB, BTS, LTS, SEL, IND, POS, OCC, PRJ, AVG, SUM, MIN, MAX } \}
\]
\[ \cup \{ \sigma \in \text{OPNS}^2 \} \]

\[ \text{PRED}_{\text{EXPR}} := \{ \text{IN} \} \]

\[ \text{OPNS}\text{ERR} := \text{ATTRIBUTE} \cup \text{COMPONENT} \]

\[ \text{PRED}\text{ERR} := \text{RELATION} \]

Please note that (for example) CNT in the above definition of \( \text{OPNS}_{\text{EXPR}} \) is short for
We can drop the indices, because no ambiguities shall occur. All operations and predicates are now concentrated to

\[
\text{OPNS} := \text{OPNS}_{\text{DS}} \cup \text{OPNS}_{\text{EE}} \cup \text{OPNS}_{\text{EXR}}
\]

\[
\text{PRED} := \text{PRED}_{\text{DS}} \cup \text{PRED}_{\text{EE}} \cup \text{PRED}_{\text{EXR}}
\]

\[
\mu[\text{SORT}], \mu[\text{OPNS}_{\text{EXR}}], \mu[\text{PRED}_{\text{EXR}}], \mu[\text{OPNS}_{\text{EE}}], \mu[\text{PRED}_{\text{EE}}], \mu[\text{OPNS}], \text{ and } \mu[\text{PRED}],
\]

are determined by the corresponding \(\mu\)'s on the right hand side.

Let us remind you that the (interpretation of the) sets DATA, ENTITY, and RELATION are disjoint as well as all (interpretations of) set-/list-/bag-constructed sort expressions (except the special value \(\bot\)). Due to the disjointness there is a unique function

\[
\text{sort} : \hat{\mu}[\text{SORT-EXPR(SORT)}] -\{\bot\} \rightarrow \text{SORT-EXPR(SORT)}
\]

yielding for an instance of \(\hat{\mu}[\text{SORT-EXPR(SORT)}]\) the sort it belongs to; the value \(\bot\) belongs to every sort. However, we shall only build sort expressions over DATA \(\cup\) ENTITY. Thus, we use the function \text{sort} only in the restricted form

\[
\text{sort} : \hat{\mu}[\text{SORTEXPRS}] -\{\bot\} \rightarrow \text{SORTEXPRS}
\]

with \(\text{SORTEXPRS} := \text{SORT-EXPR(DATA} \cup\text{ENTITY}) \cup \text{RELATION}\).

4.2 Definition: (variables and assignments)
Let a set \(\text{VAR}\) of variables and a function \(\text{type} : \text{VAR} \rightarrow \text{SORTEXPRS}\) be given. The set of assignments \(\text{ASSIGN}\) is defined by

\[
\text{ASSIGN} := \{ \alpha : \text{FUN} \mid \alpha : \text{VAR} \rightarrow \hat{\mu}[\text{SORTEXPRS} \] and \(\alpha(v) \neq \bot\) implies \(\text{type}(v)=\text{sort}(\alpha(v))\} \}
\]

The special assignment \(\varepsilon : \text{VAR} \rightarrow \{\bot\}\) is called the empty assignment. In the following \(\forall \text{VAR}\) stands for \(\forall \text{VAR}\) and \(\text{type}(v)=s\).

4.3 Remark: Our calculus has to leave the usual hierarchical structure of calculi (terms - atomic formulas - formulas), because we will allow arbitrary bag-valued terms of the form

\[
\{ t_1, \ldots, t_n \mid \delta_1 \land \ldots \land \delta_n \land \varphi \}
\]

The terms \(t_1, \ldots, t_n\) compute the target information, \(\delta_1 \land \ldots \land \delta_n\) are declarations of variables which can again use bag-valued terms of the above form to restrict the domain of a variable to a finite set (like this has been done in the second query in chapter 2).

4.4 Definition: (ranges)
The syntax of ranges is given by a set \(\text{RANGE}\) and functions \(\text{domain} : \text{RANGE} \rightarrow \text{SORTEXPRS}\) and \(\text{free} : \text{RANGE} \rightarrow F(\text{VAR})\) (\(\rho\text{-RANGE}\_s\) stands for \(\rho\text{-RANGE}\) and \(\text{domain}(\rho)=s\)).

(i) If \(s \in \text{ENTITY}\) or \(s \in \text{RELATION}\), then \(\mu[\text{RANGE}] : \text{RANGE} \rightarrow \mu[\text{SORTEXPRS}]\) and \(\mu[\text{RANGE}]_{(s,\alpha)} = \mu[\text{SORTEXPRS}]_{(s)}\).

(ii) If \(t \in \text{TERM}_{\text{set}(s)}\) with \(s \in \text{SORTEXPRS}\), then \(\mu[\text{RANGE}]_{(s,\alpha)} = \mu[\text{TERM}]_{(t,\alpha)}\).

4.5 Example: The queries in chapter 2 used the following ranges: TOWN, COUNTRY and \(\text{BTS} - \{ \text{government}(c) \mid c : \text{COUNTRY} \}\)

4.6 Definition: (declarations)
The syntax of declarations is given by a set \(\text{DECL}\) and functions \(\text{free}, \text{decl} : \text{DECL} \rightarrow F(\text{VAR})\).

(i) If \(v \in \text{VAR}, \rho_1, \ldots, \rho_n \in \text{RANGE}_{\text{type}(v)}\), and not \(\forall \text{free}(\rho_1) \ldots \text{free}(\rho_n)\), then \(\forall \rho_1 \ldots \rho_n \in \text{DECL}\),

\[
\text{free}(\forall \rho_1 \ldots \rho_n) = \text{free}(\rho_1) \ldots \text{free}(\rho_n), \text{ and } \text{decl}(\forall \rho_1 \ldots \rho_n) = \{v\}.
\]
(ii) If \( v \in \text{VAR} \), \( \varphi_1, \ldots, \varphi_n \in \text{RANGE}_{\text{type}}(v) \), \( \delta \in \text{DECL} \) with \( \text{free}(\varphi_1) \cup \cdots \cup \text{free}(\varphi_n) \subseteq \text{decl}(\delta) \), and not \( \forall \text{free}(\delta) \), then \( \forall \varphi_1, \ldots, \forall \varphi_n, \forall \delta \in \text{DECL} \), \( \text{free}(\varphi_1) \cup \cdots \cup \text{free}(\varphi_n) \cup \text{free}(\delta) \) and \( \text{decl}(\varphi_1) \cup \cdots \cup \text{decl}(\varphi_n) \cup [\delta] \cup \text{decl}(\delta) \).

The semantics of declarations is a relation \( \mu[\text{DECL}] \subseteq \text{DECL} \times \text{ASSIGN} \).

(i) \( \langle v_1 \ldots v_p, \alpha \rangle \in \mu[\text{DECL}] \) iff \( \alpha(v) \in \text{RANGE}(v_1, \alpha) \) or \( \ldots \) or \( \alpha(v) \in \text{RANGE}(v_n, \alpha) \).

(ii) \( \langle v_1 \ldots v_p, \delta, \alpha \rangle \in \mu[\text{DECL}] \) iff \( \alpha(v) \in \text{RANGE}(v_1, \alpha) \) or \( \ldots \) or \( \alpha(v) \in \text{RANGE}(v_n, \alpha) \) and \( [\delta, \alpha] \in \mu[\text{DECL}] \).

4.7 Remarks: At first, we do not only allow the form \((v; \rho)\) but also the more general \((v; \rho_1 \cup \ldots \cup \rho_n)\). This form is necessary to express every term of the relational algebra \( [\text{Ma 83}] \), especially the union of sets, in our calculus. In declarations of the form (ii) \((v_1; \rho_1) : \ldots ; (v_n; \rho_n)\), the variables \( v_1, \ldots, v_n \) (in front of the colons) are declared (each \( v_i; \rho_i \) may be a union of form (i)). Each range \( \varphi_i \) \( (i = 1 \ldots n - 1) \) may contain free variables, but only from the set of previously declared variables \( \{v_{i+1}, \ldots, v_n\} \), except \( \varphi_n \), which can have other free variables. Indeed, the later ones are the free variables of the declaration, which must be disjoint from the variables declared in the declaration.

Each variable \( v \) is bound to a finite set of values determined in the following way.

\( v_n \) can take each value from \( \mu[\text{RANGE}](\varphi_n, \alpha) \).

\( v_{n-1} \) can take each value from \( \mu[\text{RANGE}](\varphi_n, \alpha) \) "possibly dependent on the value assigned to \( v_n \) by \( \alpha \)" and so on.

4.8 Example: The queries of chapter 2 declared the following variables:

\( t : \text{TOWN} ; c : \text{COUNTRY} ; g : \text{BTS} \{ \text{government}(c) \mid c : \text{COUNTRY} \} \).

Declarations with free variable \( t \) of type \( \text{TOWN} \) are the following ones:

- "\( p : \text{LTS}(\text{geometry}(t)) \)" declares the variable \( p \) (of type \( \text{point} \)).

- "\( s : \text{streets}(d) \mid d : \text{districts}(t) \)" declares \( s \) (of type \( \text{STREET} \)) and \( d \) (of type \( \text{DISTRICT} \)).

- "\( n : \text{BTS} \{ \text{name}(r') \mid r' : \text{RIVER} \wedge r : \text{RIVER} \wedge \text{lies-at}(t, r) \wedge \text{flows-into}(r', \text{WATER}(r)) \} \)" declares \( n \) (of type \( \text{string} \)).

4.9 Definition: (terms)

The syntax of terms is given by a set \( \text{TERM} \), and functions \( \text{sort} : \text{TERM} \rightarrow \text{SORTEXPRS} \) and \( \text{free} : \text{TERM} \rightarrow F(\text{VAR}) \) (\( \text{TERM}_s \) stands for \( \text{TERM} \) and \( \text{sort}(t) = s \)).

(i) If \( v \in \text{VAR}_s \), then \( \forall \text{TERM}_s \), \( \forall \text{free}(v) := \{v\} \).

(ii) If \( v \in \text{VAR}_r \), and \( r(e_1, \ldots, e_n) \in \text{RELATION} \), then \( v.r_{\text{TERM}_s} \) \((i = 1 \ldots n) \) and \( \text{free}(v, i) := \{v\} \).

(iii) If \( c \in \text{CONSTRUCTION} \), \( s_{\text{in}} \in \text{input}(c) \), \( s_{\text{out}} \in \text{output}(c) \), \( t_{\text{in}} \in \text{TERM}_s \), and \( t_{\text{out}} \in \text{TERM}_s \),

then \( s_{\text{out}}(t_{\text{in}})_{r_{\text{TERM}_s}} \) with \( \text{free}(s_{\text{out}}(t_{\text{in}})) := \text{free}(t_{\text{in}}) \) and \( s_{\text{in}}(t_{\text{out}})_{r_{\text{TERM}_s}} \) with \( \text{free}(s_{\text{in}}(t_{\text{out}})) := \text{free}(t_{\text{out}}) \).

(iv) If \( \omega \in \text{OPNS} \) and \( t_{\text{TERM}_s} \) \((i = 1 \ldots n) \), then \( \omega(t_{\text{term}_1}, \ldots, t_{\text{term}_n})_{r_{\text{TERM}_s}} \) with \( \text{free}(\omega(t_{\text{term}_1}, \ldots, t_{\text{term}_n})) := \text{free}(t_{\text{term}_1}) \cup \cdots \cup \text{free}(t_{\text{term}_n}) \).

(v) If \( t, t_{\text{TERM}_s} \) \((i = 1 \ldots n) \), \( \delta \in \text{DECL} \) \((j = 1 \ldots k) \), \( \varphi \in \text{FORM} \), and \( \text{dec}(\delta) \cup \text{decl}(\delta) = \text{dec}(\delta) \cup \text{free}(\delta) = \emptyset \) \((i, j)\),

then \( t \mid \cdots \mid t_n \mid \delta_1 \wedge \cdots \wedge \delta_k \wedge \varphi \rightarrow_{\text{TERM}} t_{\text{TERM}_s} \) (prod \((t_{\text{term}_1}, \ldots, t_{\text{term}_n})\)) and \( \text{free}(t_{\text{term}_1}) \cup \cdots \cup \text{free}(t_{\text{term}_n}) \cup \text{free}(\delta) \).

The semantics of terms is a function \( \mu[\text{TERM}] : \text{TERM} \times \text{ASSIGN} \rightarrow [\mu]\text{SORTEXPRS} \).

(i) \( \mu[\text{TERM}](v, \alpha) := \alpha(v) \).

(ii) \( \mu[\text{TERM}](v, i, \alpha) := \text{if } \mu[\text{TERM}](v, \alpha) = (x_1, \ldots, x_n) \text{ then } x_i \text{ else } 1 \).
\[ \mu(\text{TERM})\{s\text{out}(t_{in}),\alpha\} := \begin{cases} \text{if there is } e \in \text{ENTITY}(s\text{out}) : \mu(\text{CONS.})(c)(e) = \mu(\text{TERM})(t_{in}, \alpha) & \text{then } e \text{ else } \bot. \\
\mu(\text{TERM})\{s\text{in}(t_{out}),\alpha\} := \begin{cases} \text{if } \mu(\text{CONS.})(c)(\mu(\text{TERM})(t_{out}, \alpha)) = \mu(\text{ENTITY})(s\text{in}) & \text{then } \mu(\text{CONS.})(c)(\mu(\text{TERM})(t_{out}, \alpha)) \text{ else } \bot. \\
\end{cases} \\
\end{cases} \]

(v) \[ \mu(\text{TERM})(\omega(t_1,...,t_n),\alpha) = \mu(\text{CONS.})(c)(\mu(\text{TERM})(t_1,\alpha),...,,\mu(\text{TERM})(t_n,\alpha)). \]

4.10 Remark: If no ambiguity arises, we use entity types instead of integers in the case of terms according to point (ii), e.g., ft.RIVER instead of ft.1 (with type(ft)=flows-through).

4.11 Example: The queries in chapter 2 are term according to point (vi) of the above definition and have sort bag(prod{string r at}) and bag(prod{string int}), respectively.

- \{ population(c) | c : COUNTRY ∧ government(c)=g \} is a terms of sort bag(int) with free variable g. It was used in the second query in chapter 2.

- \{ r | r : RIVER ∧ lies-in(r,c) \} is a term of sort bag(RIVER) with free variable c.

4.12 Definition: (formulas)

The syntax of formulas is given by a set FORM and a function free : FORM → F(VAR).

(i) If \(\pi; t_1,...,t_n \in \text{PRED}\) and \(t_i \in \text{TERM} \ (i=1,...,n)\), then \(\pi(t_1,...,t_n) \in \text{FORM}\) and \(\text{free}(t_1,...,t_n) := \text{free}(t_1) \cup ... \cup \text{free}(t_n)).\)

(ii) If \(t_1,t_2 \in \text{TERM}\), then \(t_1=t_2 \in \text{FORM}\) and \(\text{free}(t_1,t_2) := \text{free}(t_1) \cup \text{free}(t_2).\)

(iii) If \(t \in \text{TERM}\), then \(\text{UNDEF}(t) \in \text{FORM}\) and \(\text{free}(\text{UNDEF}(t)) := \text{free}(t).\)

(iv) If \(\varphi \in \text{FORM}\), then \(\neg(\varphi) \in \text{FORM}\) and \(\text{free}(\neg(\varphi)) := \text{free}(\varphi).\)

(v) If \(\varphi_1, \varphi_2 \in \text{FORM}\), then \((\varphi_1 \lor \varphi_2) \in \text{FORM}\) and \(\text{free}(\varphi_1 \lor \varphi_2) := \text{free}(\varphi_1) \cup \text{free}(\varphi_2).\)

(vi) If \(\varphi \in \text{FORM}\) and \(\delta \in \text{DECL}\), then \(\exists(\varphi) \in \text{FORM}\) and \(\text{free}(\exists(\varphi)) := (\text{free}(\varphi) \cup \text{decl}(\delta)) \cup \text{free}(\delta).\)

The semantics of formulas is a relation \(\mu(\text{FORM}) \subseteq \text{FORM} \times \text{ASSIGN}.\)

(i) If \(\pi; t_1,...,t_n \in \text{FORM}\), then \(\pi(t_1,...,t_n) \in \mu(\text{FORM} \times \text{ASSIGN})).\)

(ii) If \(t_1=t_2 \in \text{FORM}\), then \(\mu(\text{TERM})(t_1,\alpha) = \mu(\text{TERM})(t_2,\alpha).\)

(iii) If \(\text{UNDEF}(t,\alpha) \in \text{FORM}\), then \(\mu(\text{TERM})(t,\alpha) = 1.\)

(iv) If \(-(\varphi,\alpha) \in \text{FORM}\), then \(-\varphi(\varphi,\alpha) \in \mu(\text{FORM}).\)

(v) \((\varphi_1,\varphi_2,\alpha) \in \text{FORM}\), then \(\mu(\varphi_1,\varphi_2,\alpha) \in \mu(\text{FORM})\text{ or } (\varphi_2,\alpha) \in \mu(\text{FORM}).\)

(vi) If \(\exists(\varphi,\alpha) \in \text{FORM}\), then \(\exists(\varphi,\alpha) \in \text{ASSIGN}\) with \(\exists(\varphi,\alpha) = \alpha(\nu)\) for \(\nu \in \text{VAR} \cup \text{decl}(\delta)\)

and \((\delta,\alpha) \in \mu(\text{DECL})\).

4.13 Remark: We also use the other logical connectives ∧, →, xor, etc. and the quantifier ∃ with the usual semantics. They can be defined by means of the above definitions.

4.14 Definition: (queries)

The syntax of queries is the set QUERY:

- If \(t \in \text{TERM}\) with \(\text{sort}(t) \in \text{SORT-EXPR}(\text{DATA})\) and \(\text{free}(t) = \emptyset\), then \(t \in \text{QUERY}\).

The semantics of queries is a function \(\mu(\text{QUERY}) : \text{QUERY} \rightarrow \hat{\mu}(\text{SORT-EXPR}(\text{DATA})).\)

- \(\mu(\text{QUERY})(t) := \mu(\text{TERM})(t,\alpha).\)
4.15 Facts:
- Every multi-valued term \( t \in \text{TERM} \) is evaluated to a finite set, bag, or list for every assignment \( \alpha \).
- Since a query is a special case of a term, any query yields a finite result: The EER calculus is safe.
- The EER calculus is relationally complete (if every relation is modelled by an entity type). The proofs can be found in [HG 88]. Thus our calculus preserves nice properties of the relational calculi [Ma 83], but on the other hand it is also more expressive. In the classical calculi it is not possible to compute for instance the cardinality (i.e., the number of actually stored tuples) of a relation \( R \ (a_1 : D_1, ..., a_n : D_n) \). Of course, this presents no problem in our approach, because we can use data type operations \( \text{CNT}_{\text{bag}(R)} \{ r \mid r : R \} \). We here assume the relation \( R \) is modelled by an entity type \( R \) with attributes \( a_1, ..., a_n \).

5. Integrity Constraints

5.1 Concept: We now use our calculus to formulate integrity constraints. An integrity constraint is a formula of the calculus without free variables. These formulas restrict the class of EER-algebras to algebras where the formulas hold.

5.2 Definition: (functional restriction)
The syntax of a functional restriction for a relation \( r(e_1, ..., e_n) \) is a pair \((f = f_1, ..., f_m, g = g_1, ..., g_k)\) of subsets of \( e_1, ..., e_n \). The semantics of a functional restriction is the following formula.

\[
( \forall v, w : r ) \quad \forall f_1 = w.f_1 \land ... \land \forall f_m = w.f_m \Rightarrow \forall g_1 = w.g_1 \land ... \land \forall g_k = w.g_k \quad \Box
\]

5.3 Remark: Please notice, it is possible to formulate more than one functional restriction for a single relationship. For instance, if we have a relationship \( r(A,B,C) \) with three entity types, one can demand that \( r \) is a function from \( A \) to \( C \) as well as from \( C \times B \) to \( A \).

5.4 Example: In the example in chapter 2 the functional restriction \( ([\text{RIVER}],[\text{WATERS}]) \) on the relationship flows-into was given. It also makes sense to require the functional restriction \( ([\text{TOWN}],[\text{COUNTRY}]) \) for lies-in in order to express that a town lies in exactly one country.

5.5 Definition: (key specification)
The syntax of a key specification for an entity type \( e \) is given by a subset \( a_1, ..., a_n \) of the attributes of \( e \), a subset \( c_1, ..., c_m \) of the components of \( e \) and a subset \( r_1, ..., r_m \) of the relationships of \( e \). The semantics of a key specification is given by the formula

\[
( \forall v, w : e ) \quad v \neq w \Rightarrow \\
\left[ \begin{array}{c}
\forall a_1(v) \neq a_1(w) \lor \ldots \lor a_n(v) \neq a_n(w) \lor \\
\forall c_1(v) \neq c_1(w) \lor \ldots \lor c_m(v) \neq c_m(w) \lor \\
\left( \exists s_1 \right) r_1(v, s_1) \lor r_1(w, s_1) \lor \ldots \lor \left( \exists s_m \right) r_m(v, s_m) \lor r_m(w, s_m) \right]
\]

5.6 Remarks: The \( \ast \)-notation has to be explained shortly: Suppose \( r(A,B,C) \) is given. Then the relationship \( r \) is part of a key for the entity type \( B \), if the following formula holds.
Please notice, the other direction of the implication in the key specification formula trivially holds. Thus, key specifications can be regarded as characterizations of equality. Key relations also allow more subtle objects than key functions [EDG 86]. Consider the following simple example:

Suppose the data type d has only two values 0 and 1, the attribute a is the only key for E1 and the relationship r the only one for E2. Because the attribute a can only take two values there are (up to isomorphism) only two possible entities e1 and e2 of type E1, which can be observed to be inequivalent: a(e1)=0 and a(e2)=1. In the case of a key function r from E2 to E1 there are also only two observable inequivalent entities for E2. But, if r is an arbitrary relation, there are (up to isomorphism) four possible entities for E2: e21 related with no E1-entity, e22 related with e1, e23 related with e1 and e24 related with e1 and e2. It is an open problem how to construct "universes" [EDG 86, SSE 87] for key relations. The above example suggests some kind of power set construction yielding a final algebra analogously to [EDG 86, SSE 87].

5.7 Example: The example in chapter 2 required that \{name,flows-through\} is a key for RIVER. For the entity type TOWN one can demand \{name,lies-in\} as well as \{geometry\} to be a key:

\[(\forall t, t': \text{TOWN})\]
\[t \neq t' \Rightarrow \text{name}(t) \neq \text{name}(t') \vee (\exists c: \text{COUNTRY}) \text{lies-in}(t,c) \text{xor} \text{lies-in}(t',c)\]
\[\wedge\]
\[t \neq t' \Rightarrow \text{geometry}(t) \neq \text{geometry}(t')\]

The last line says: If two towns have the same geometry (the same list of points representing the town's border), then the towns are identical. This example shows that more than one key for a single entity type can make sense.

5.8 Definition: (cardinality constraint)
The syntax of a cardinality constraint for a set-, list- or bag-valued attribute or component f of entity type e is given by a pair of integers (low,high). The semantics is given by

\[(\forall v : e) \text{ low} \leq \text{CNT}(f(v)) \land \text{CNT}(f(v)) \leq \text{high}.\]

5.9 Example: The example in chapter 2 restricted the number of streets for a district. These kind of constraints cannot be expressed in the "classical" relational calculi. Another such typical example again involves aggregate functions. Domain or tuple calculus cannot express a constraint like "The average age of the ministers of a country should not be greater than 65". If there is a component "ministers : \text{COUNTRY} \rightarrow \text{set} (\text{PERSON})" and an attribute "age : \text{PERSON} \rightarrow \text{int}". This constraint can be expressed in our calculus like

\[\forall c: \text{COUNTRY} \quad (\text{AVG} \{\text{age}(p) \mid p: \text{PERSON} \land p \in \text{ministers}(c)\} \leq 65).\]

Acknowledgement

Thanks go to Uwe Hohenstein for our collaboration on the model and the calculus. Special acknowledgement goes to Yugoslavia: For the kindness of the people of "Starigrad Paklenica" and for the uniqueness of the "Plitvička Jezera" region.
References


[Bü 87] Bültzingsloewen, G. v.: Translating and Optimizing SQL Queries Having Aggregates. 13th VLDB, Brighton 1987


[ÖÖM 87] Özsoyoglu, G. / Özsoyoglu, Z.M. / Matos, V.: Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. ACM Transactions on Database Systems, Vol. 12, No. 4, December 1987 (pp. 566-592)


