An FCA interpretation of Relation Algebra*

Uta Priss

School of Computing, Napier University, Edinburgh, UK
www.upriss.org.uk
u.priss@napier.ac.uk

Abstract. This paper discusses an interpretation of relation algebra and fork algebra with respect to FCA contexts. In this case, "relation algebra" refers to the DeMorgan-Peirce-Schroeder-Tarski algebra and not to the "relational algebra" as described by Codd. The goal of this interpretation is to provide an algebraic formalisation of object-relational databases that is based on binary relations and thus closer to FCA and formal contexts than the traditional formalisation based on Codd. The formalisation provides insights into certain symmetries (among quantifiers) and the use of ternary relations and part-whole relations for building relational databases.

1 Introduction

Algebras of relations, such as Codd's (1970) relational algebra (RLA) or Peirce-Tarski's relation algebra (RA)[1], have been studied by logicians since the mid 19th century. But apart from the use of RLA in relational databases, relational methods have not been in the mainstream for more than a hundred years, even though they have promising applications. Only during the past 15 years, there has been an increased interest in "Relational Methods in Computer Science" as evidenced by the creation of a new journal in this area[2].

Relational methods can be considered a "paradigm" that is different from some set-based logical formulas because a relational representation abstracts from elements and certain quantifiers. Programming languages that are based on relational methods tend to be more of a non-functional, list processing character. Users sometimes find such languages or formalisms difficult to read - as has been documented with respect to the relational database language SQL (eg. Hansen & Hansen (1988)). This may explain why relational methods are only slowly gaining more popularity. Nevertheless, relational methods have interesting applications and because of the recent interest in relational methods in computer science and because RA and FCA share common structures, we believe that a detailed discussion of FCA and RA is of interest to the FCA community and provides links to this newly emerging research area.

* This is a preprint of a paper published in Missaoui; Schmidt (eds.), Formal Concept Analysis: 4th International Conference, ICFCA 2006, Springer Verlag, LNCS, 2006, p. 248-263. ©Springer Verlag.

[1] In this paper "RLA" is used as an abbreviation for Codd's relational algebra and "RA" for a Tarski-style relation algebra.

[2] http://www.jormics.org

A combination of RA and FCA can be used to analyse formal aspects that underpin relational and object-relational databases. Current RLA-based implementations of databases are highly optimised with respect to functionality and efficiency. But RA can provide new insights into the structural properties of relational databases, such as into certain symmetries (among quantifiers) and the use of ternary relations and part-whole relations for relational databases. Currently, there does not exist a widely accepted extension of RLA to object-relational databases (cf. Atkinson et al. (1989)). A broader approach using a variety of algebras (including RA and RLA) may lead to such a formalisation of object-relational databases and formal ontologies. With respect to FCA, this paper shows that RA is sufficiently expressive to represent basic FCA notions and a fork extension of RA is sufficient to represent many-valued contexts and power contexts.

This paper presents a continuation and elaboration of some ideas which were presented in a preliminary form by Priss (2005). But in contrast to Priss (2005), this paper adds a more detailed mathematical presentation, a use of fork algebras, a distinction between a "named" and an "unnamed" perspective and a more detailed elaboration of relational schemata.

2 Algebras of Relations: Codd versus Tarski

The most influential algebra that is currently used in computer science is probably RLA because it serves as the foundation of relational databases. It is not a trivial task to mathematically formalise RLA in detail with respect to relational databases - as indicated by the fact that at least four different types of suggestions for such formalisations of RLA exist (Abiteboul et al., 1995). Because of this and because RLA uses n-ary relations, it is also not trivial to combine relational databases directly with Formal Concept Analysis (FCA). Wille's (2002) notion of power context families incorporates n-ary relations into FCA, but it does not cover all the detail of relational databases and RLA. Hereth (2002) has made some progress with respect to RLA and FCA.

Although Codd (1970) is usually quoted as the inventor of RLA (and he certainly advocated the practical use of it), a more detailed and comprehensive description of algebras of relations was provided by Tarski in the 1940s (cf. Van den Bussche (2001) for an overview). Tarski described two types of algebras: RA and Cylindric Set Algebra, which according to Imielinski & Lipski (1984) is closely connected to RLA. The idea of RA can be traced back from Tarski (1941) to Peirce and de Morgan and Schröder (cf. Pratt (1992) for an overview). In contrast to RLA which has expressive power equivalent to first order logic, the expressive power of RA is only equivalent to first order logic with at most 3 distinct variables (cf. Van den Bussche (2001)). Thus RA is much less powerful than RLA. But there is an extension of RA called "fork algebra", which is equivalent to first order logic. Because of a close relationship between RA and binary relations, it is of interest to consider RA and fork algebra together with FCA.

3 Relation Algebra: Definition and Overview

The following definition follows Tarski and is adapted from Brink et al. (1992):

Definition 1 A *relation algebra* is an algebra $(R, +, \cdot, ', 0, 1, ;, ^\smile, e)$ satisfying the following axioms for each $r, s, t, \in R$:

R1	$(R, +, \cdot, ', 0, 1)$ is a Boolean algebra	R5	$(r + s); t = r; t + s; t$
R2	$r; (s; t) = (r; s); t$	R6	$(r + s)^\smile = r^\smile + s^\smile$
R3	$r; e = r = e; r$	R7	$(r; s)^\smile = s^\smile; r^\smile$
R4	$r^{\smile\smile} = r$	R8	$r^\smile; (r; s)' \leq s'$.

If R in Definition 1 is a set of binary relations, then the following can be defined:

Definition 2 A *proper relation algebra* (RA) is an algebra $(R, \cup, ^-, one, \circ, ^d, dia)$ where for a set A and an equivalence relation $one \subseteq A \times A$, R is a set of binary relations equal to the powerset of one and $dia := \{(x, x) \in one\}$ and for $I, J \in R$:
$I \cup J := \{(x, y) \mid (x, y) \in I \text{ or } (x, y) \in J\}$; $\overline{I} := \{(x, y) \mid (x, y) \in one, (x, y) \notin I\}$;
$I \circ J := \{(x, y) \mid \exists_{z \in A} : (x, z) \in I \text{ and } (z, y) \in J\}$; $I^d := \{(x, y) \mid (y, x) \in I\}$.

Table 1. Overview of basic RA operations and some extensions

RA	Tarski's	name	basis	definitions
$\cup$	$+$	union	yes	
$^-$	$'$	negation (complement)	yes	
$\cap$	$\cdot$	intersection		$I \cap J := \overline{\overline{I} \cup \overline{J}}$
$\circ$	$;$	composition	yes	
^{d}	$^\smile$	inverse (dual)	yes	
$\bullet$	$\dagger$	de Morgan compl. of $\circ$		$I \bullet J := \overline{\overline{I} \circ \overline{J}}$
nul	0			$nul := \overline{one}$
one	1	universal relation	yes	
dia	e	diagonal	yes	
$\subseteq$				$I \subseteq J :\Leftrightarrow I \cap J = I$
$=$		equality		$I = J :\Leftrightarrow I \subseteq J$ and $J \subseteq I$
$\subset$		containment		$I \subset J :\Leftrightarrow I \subseteq J$ and not $I = J$
^{trs}		transitive closure	(yes)	$I^{trs} := I \cup I \circ I \cup I \circ I \circ I \cup \ldots$
*		refl. trans. closure		$I^\star := dia \cup I^{trs}$

Different authors use different symbols for RA operations. It is common to list the Boolean operators before the non-Boolean ones in the signature of the algebra. It is also common to use Tarski's notation $(+, \cdot, ;, 0, 1)$ for relation algebras and $\cup, \cap, ^-$ and other symbols for proper relation algebras. The left column of table 1 shows the notations used in this paper. For the purposes of this paper, a mapping from an equational class (as in definition 1) to an algebra which has elements with set-theoretically defined structure (as in definition 2) is called an "interpretation". Apart from interpreting relation algebras with respect to binary relations, they can also be interpreted with respect to FCA contexts, as shown in later sections of this paper. Pratt (1993) observes that only the non-Boolean operations $(\circ, ^d)$ make use of the inner structure of the elements of the relations (such as inverting the pairs using ^{d}). For the Boolean operations $(\cup, \cap, ^-)$, relations are just sets. It has been shown by Lyndon (1950) that there are interpretations of relation algebras which are not isomorphic to proper relation algebras. In this paper, only proper relation algebras are considered and their notational symbols are used. "RA" stands for proper relation algebra in the remainder of this paper. Based on defi-

nition 2, a representable relation algebra (RRA) is usually defined as a subalgebra of a proper relation algebra. The class RRA forms a variety (Tarski, 1955).

The top half of table 1 shows how other common operations and elements of RA can be derived from the basis operations and elements. Numerous mathematical (or logical) properties can be proven for RA (cf. Maddux (1996), Pratt (1992), Pratt (1993), Van den Bussche (2001), Kim (1982)). There are numerous applications for RA, which obviously include and extend applications of Boolean algebras. Apart from applications in logic, RA has been used for the semantics of programming languages (Maddux, 1996). Pratt (1993) explains that RA is very similar to Chu spaces.

The bottom half of table 1 shows some extensions of RA: equality and transitive closure. For this paper, equality and containment is assumed to be defined for RA. Transitive closure is not a first order logic property and cannot be derived from the other RA operations. It can be useful in some applications to have transitive closure available. For example, if I represents the incidence matrix of a graph, then I^{trs} shows all transitive paths between the graph nodes. We believe that a major reason for the recent popularity of XML for ontologies and other tree-like structures is because the calculation of paths in a tree is natural for XML but difficult in SQL. In fact, only the more recent SQL standard (SQL3) contains a suggestion for a recursion operator that can be used for calculating paths in a tree. Unfortunately, the implementation of this operator is inconsistent among different database vendors (Wagner, 2003). The reasons for this may be that transitive closure is missing from RLA and that it can be computationally expensive to calculate transitivity. Nevertheless, we believe, that if transitive closure had been added to SQL at an earlier stage, the history of XML as a format for representing ontologies might have been different. This brief excursus on XML and databases should indicate the significance of the presence or absence of a transitive closure operation. In this paper, we assume that transitive closure is available as needed.

4 RA Interpretations as FCA Contexts

4.1 Active Domains

In analogy to relational database theory, an "active domain" (`ACT`) is introduced for the purposes of this paper. In relational database theory, an active domain is the finite set of actually occurring values and value combinations, which is a subset of the infinite "universe" (`U`) of possible values. For example, the complement of a relation in relational databases is usually calculated with respect to the active domain to avoid the use of infinite sets. Relational databases contain finite sets of data at any point in time, but a fork operation as introduced in section 5.2 requires an infinite set of elements at all times. To cope with the infinity of the fork operation, the following two sets are defined in this paper: a set of identifiers containing a finite set of even-numbered identifiers (`EVN`) and an infinite set of odd-numbered identifiers (`ODD`). Even-numbered identifiers are used for actual, persistent or "important" data and odd-numbered ones are used for potential, transient or "un-important" data. This distinction follows the practise of object-relational databases which automatically generate "object identifiers" for instances of tables. It also follows the distinction between "persistent" data (for data that may need

to be reused in other applications and should be stored) and "transient" data (which can be forgotten, such as the values of a counter) by Atkinson et al. (1989). It should be noted that even though the set of even identifiers is finite and fixed at any point in time, it can change over time if new data is added to an application (eg. if database tables are updated). In addition to the sets of identifiers, there is also a finite set N of named elements of a relational database (i.e., names of tables, columns, values, etc).

Definition 3 A *universe* of possible elements is a set $\mathtt{U} := \mathtt{N} \cup \mathtt{EVN} \cup \mathtt{ODD}$ where N is a finite set of names, EVN is a finite set of even-numbered identifiers and ODD is an infinite set of odd-numbered identifiers and $\mathtt{N}, \mathtt{EVN},$ and ODD are pairwise disjoint. An *active domain* is a finite subset of U defined as $\mathtt{ACT} := \mathtt{N} \cup \mathtt{EVN}$. For practical purposes, ACT is assumed to have a fixed linear order.

4.2 The unnamed perspective

There are potentially numerous ways for using RA with respect to FCA. Because formal contexts are usually represented as cross tables, for the rest of this paper binary relations are viewed as Boolean matrices (or binary matrices or cross-tables) in the sense of Kim (1982). In analogy to a distinction made in relational database theory (Abiteboul, 1995), we distinguish between an "unnamed perspective" and a "named perspective". In the unnamed perspective, all data of an application, i.e., all formal contexts of an application, are represented as (possibly large) matrices of the same dimension[3]:

Definition 4 In the unnamed perspective, with $|\mathcal{A}| := |\mathtt{ACT}| \times |\mathtt{ACT}|$, an *active domain* $\mathcal{A}$ is the set of all binary $|\mathcal{A}|$-dimensional matrices I so that semantically for all elements in ACT, the nth element in ACT corresponds to the nth row and column in I. It is then said that I *is based on* $\mathcal{A}$ denoted by $I_{\mathcal{A}}$.

The subscript $\mathcal{A}$ in $I_{\mathcal{A}}$ can be omitted if it clear from context. Obviously, it would be impractical for most applications to actually construct such large matrices. The unnamed perspective is mainly used to define some operations in a somewhat more context-independent manner, which can be useful for certain context compositions in the named perspective. Otherwise, the unnamed perspective is mostly of theoretical value. Each object or attribute of any formal context relating to a single application is uniquely identified by its position in $\mathcal{A}$. Row and column permutations change the values of rows and columns but not their semantic correspondence to elements in $\mathcal{A}$ (thus may not be meaningful operations). Even though, the names of the elements in ACT are not strictly required, it is usually more convenient to use them instead of using row and column numbers.

The next definition assumes the usual operations for Boolean matrices (cf. Kim (1982)), i.e., with $(i,j)_I$ denoting the element in row i, column j in matrix I and $\vee$, $\wedge$ and $\neg$ denoting Boolean OR, AND and NOT: $(i,j)_{I \cup J} := (i,j)_I \vee (i,j)_J$; $(i,j)_{\overline{I}} := \neg(i,j)_I$; $(i,j)_{I \circ J} := 1$ iff $\exists_k : (i,k)_I \wedge (k,j)_J$; $(i,j)_{I^d} := (j,i)_I$. A matrix is symmetric if $I = I^d$, reflexive if $dia \subseteq I$, transitive if $I^2 \subseteq I$. Because matrix

[3] In the rest of this paper, typewriter font (A, B, etc) is used for subsets and elements of ACT and uppercase italics (I, J etc) for matrices and binary relations. If elements of ACT are used in names of matrices, then they are written in italics but underlined (but not if used as subscript).

operations and operations on binary relations are so similar, we use a set-theoretic notation for both. The distinction between sets and matrices is made using typeface (see footnote 3).

Definition 5 A *matrix-RA based on* $\mathcal{A}$ is an algebra $(R, \cup, ^-, one, \circ, ^d, dia())$ where $one \in R$ is a reflexive, symmetric and transitive matrix; R is a set of Boolean matrices based on $\mathcal{A}$ with $I \in R \Leftrightarrow I \subseteq one$; and $\cup, ^-, \circ, ^d$ are the usual Boolean matrix operations; and for any set $\mathtt{S} \subseteq \mathtt{ACT}$ and $\mathtt{a(n)}$ denoting the nth element in $\mathtt{ACT}$, $dia(\mathtt{S})$ is defined by $(i,j)_{dia(\mathtt{S})} = 1$ iff $i = j$ and $\mathtt{a(i)} \in \mathtt{S}$ (but only if $dia(\mathtt{S}) \subseteq one$).

Table 2 summarises the definition and introduces some further operations (with $\mathtt{G}, \mathtt{M}, \mathtt{S} \subseteq \mathtt{ACT}$). The operation $\cap$ is still the de Morgan complement of $\cup$. nul can be derived from dia or via $nul = \overline{one}$. Because binary relations can be equivalently represented as sets of pairs or as binary matrices it follows that:

Lemma 1 Definitions 2 and 5 are equivalent: every RA is a matrix-RA and vice versa.

It is not necessary to use sets ($\mathtt{S}, \mathtt{ACT}$) in definition 5. Instead of defining $dia(\mathtt{S})$, one could define dia and then derive $dia(I_{\rightarrow}), dia(I_{\uparrow})$ and state that every matrix $J \subseteq dia$ corresponds to a set. Thus the algebra in definition 5 is not truly two sorted. But the use of $dia(\mathtt{S})$ is convenient with respect to the named perspective. The other definitions from table 2 can be explained as follows: the matrices $dia(I_{\uparrow})$ and $dia(I_{\rightarrow})$ represent column-wise and row-wise projections of a matrix I onto the diagonal. For $dia(I_{\uparrow})$ this means that for each column in I that contains at least one 1, $dia(I_{\uparrow})$ contains a 1 in that position on the diagonal. A matrix $sqr(\mathtt{G}, \mathtt{M})$ contains a 1 for each cell whose row name is in $\mathtt{G}$ and whose column name is in $\mathtt{M}$. $sqr(\mathtt{G}, \mathtt{M})$ is an encoding of an empty cross table of a formal context based on $\mathcal{A}$. A formal context can now be represented as $(sqr(\mathtt{G}, \mathtt{M}), I)$ where $\mathtt{G}, \mathtt{M} \subseteq \mathtt{ACT}$ and I is a matrix based on $\mathcal{A}$ with $I \subseteq sqr(\mathtt{G}, \mathtt{M})$.

Table 2. The unnamed perspective: A matrix-RA based on $\mathcal{A}$

notation	definition	basis
$\cup$	component-wise $\vee$	yes
$^-$	component-wise $^-$	yes
$\circ$	binary matrix multiplication	yes
^{d}	matrix transposition (mirrored along diagonal)	yes
nul	$:= \overline{dia \cup \overline{dia}}$	
dia	$:= dia(\mathtt{ACT})$	
$dia(\mathtt{S})$	has 1's according to $\mathtt{S}$	yes
$dia(I_{\uparrow})$	$:= dia \cap (one \circ I) = I^d \circ I \cap dia$	
$dia(I_{\rightarrow})$	$:= dia \cap (I \circ one) = I \circ I^d \cap dia$	
$sqr(\mathtt{G}, \mathtt{M})$	$:= dia(\mathtt{G}) \circ one \circ dia(\mathtt{M})$	

Definition 6 A *context-RA based on* $\mathcal{A}$ for a set of formal contexts is the smallest matrix-RA based on $\mathcal{A}$ that contains these contexts.

This means that for a context $(sqr(\mathtt{G}, \mathtt{M}), I)$ the context-RA contains all contexts that have any subsets of $\mathtt{G} \cup \mathtt{M}$ as sets of objects and attributes. It should be noted that a smaller RRA could be constructed that contains a set of contexts, if R in definition 5 was not required to contain all matrices $I \subseteq one$. But since the prime operator ($'$) in FCA is

normally applicable to all subsets of objects or attributes, definition 5 (which allows the formation of matrices corresponding to subsets) seems reasonable. Subsets of $\mathtt{G}$ and $\mathtt{M}$ can be represented as diagonal matrices or as matrices which contain identical rows (eg. $sqr(\mathtt{ACT}, \mathtt{S})$) or identical columns (eg. $sqr(\mathtt{S}, \mathtt{ACT})$). These three ways are equivalent because the matrices can be converted: $dia(\mathtt{S}) = sqr(\mathtt{ACT}, \mathtt{S}) \cap dia = sqr(\mathtt{S}, \mathtt{ACT}) \cap dia$ and $sqr(\mathtt{ACT}, \mathtt{S}) = one \circ dia(\mathtt{S})$.

Lemma 2 In the unnamed perspective the basic FCA operations can be represented as summarised in table 3.

Table 3. Basic FCA operations in the unnamed perspective

standard FCA	RA: unnamed perspective
$\mathtt{g} I \mathtt{m}$	$sqr(\{\mathtt{g}\}, \{\mathtt{m}\}) \subseteq I$
$\mathtt{g}' := \{\mathtt{m} \in \mathtt{M} \mid \mathtt{g} I \mathtt{m}\}$	$dia(\underline{g}') = dia(\underline{g}^+) := dia \cap sqr(\mathtt{ACT}, \{\mathtt{g}\}) \circ I$
$\mathtt{H}' := \{\mathtt{m} \in \mathtt{M} \mid \forall_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \Longrightarrow \mathtt{g} I \mathtt{m}\}$	$dia(H') = dia \cap \overline{sqr(\mathtt{ACT}, \mathtt{H}) \circ \overline{I}}$
$\mathtt{H}^+ := \{\mathtt{m} \in \mathtt{M} \mid \exists_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \text{ and } \mathtt{g} I \mathtt{m}\}$	$dia(H^+) = dia \cap sqr(\mathtt{ACT}, \mathtt{H}) \circ I$

The equivalence of the expressions in the left and right columns in table 3 follows directly from the definitions. But a further explanation of the table is required: a context $(sqr(\mathtt{G}, \mathtt{M}), I)$ is assumed with $\mathtt{H} \subseteq \mathtt{G}$; $\mathtt{N} \subseteq \mathtt{M}$; $\mathtt{g} \in \mathtt{G}$; $\mathtt{m} \in \mathtt{M}$. The plus ($+$) operator, which is somewhat dual to the prime ($'$) operator originates from use in lexical databases (cf. Priss(1998) and Priss & Old (2004)). The operations for sets of attributes are analogous to the ones for sets of objects in table 3.

4.3 The named perspective

In contrast to the unnamed perspective where all matrices of an application are of dimension $|\mathcal{A}|$, in the named perspective matrices can have different dimensions and may not even be square.

Definition 7 In the named perspective, the *active domain* ACT is linearly ordered. A formal context $(\mathtt{G}, \mathtt{M}, I)$ *based on* ACT consists of two sets $\mathtt{G}, \mathtt{M} \subseteq \mathtt{ACT}$, which are linearly ordered using the by $\mathtt{ACT}$-induced ordering, and of a binary matrix I of dimension $|\mathtt{G}| \times |\mathtt{M}|$ where the ith row corresponds to the ith element in $\mathtt{G}$ and the jth column corresponds to the jth element in $\mathtt{M}$. This can be denoted as $I_{\mathtt{G},\mathtt{M}}$.

Semantically, this implies a unique name assumption because if the same name is used in different formal contexts or in a single context both as an object and as an attribute, then these elements are semantically indistinguishable because they refer to the same element in $\mathtt{ACT}$. The unique name assumption ensures that the operations $\cup$ and $\circ$ can be meaningfully generalised to contexts of different dimensions as follows:

Definition 8 For formal contexts $\mathcal{K}_1 := (\mathtt{G}_1, \mathtt{M}_1, I)$ and $\mathcal{K}_2 := (\mathtt{G}_2, \mathtt{M}_2, J)$ the following *context operations* are defined:
$\mathcal{K}_1 \sqcup \mathcal{K}_2 := (\mathtt{G}_1 \cup \mathtt{G}_2, \mathtt{M}_1 \cup \mathtt{M}_2, I \sqcup J)$ with $\mathtt{g} I \sqcup J \mathtt{m} :\Longleftrightarrow \mathtt{g} I \mathtt{m}$ or $\mathtt{g} J \mathtt{m}$
$\mathcal{K}_1 \sqcap \mathcal{K}_2 := (\mathtt{G}_1 \cup \mathtt{G}_2, \mathtt{M}_1 \cup \mathtt{M}_2, I \sqcap J)$ with $\mathtt{g} I \sqcap J \mathtt{m} :\Longleftrightarrow \mathtt{g} I \mathtt{m}$ and $\mathtt{g} J \mathtt{m}$
$\mathcal{K}_1 \diamond \mathcal{K}_2 := (\mathtt{G}_1, \mathtt{M}_2, I \diamond J)$ with $\mathtt{g} I \diamond J \mathtt{m} :\Longleftrightarrow \exists_{\mathtt{n} \in (\mathtt{M}_1 \cap \mathtt{G}_2)} : \mathtt{g} I \mathtt{n}$ and $\mathtt{n} J \mathtt{m}$
$\overline{\mathcal{K}_1} := (\mathtt{G}_1, \mathtt{M}_1, \overline{I})$; $\mathcal{K}_1^d := (\mathtt{M}_1, \mathtt{G}_1, I^d)$.

Table 4 shows some further operations that can be defined for formal contexts in the named perspective. Most of the operations are essentially the same as in the unnamed perspective. Because dia is square, one set as subscript is sufficient ($dia_{\mathtt{G}} := dia_{\mathtt{G},\mathtt{G}}$). In addition to operations which convert a set into a matrix ($dia_{\mathtt{M}}(\mathtt{S}_{\rightarrow})$), there are also operations which convert a matrix into a set: $\mathtt{set}_{\mathtt{G}}(I)$. Union potentially enlarges the dimension of the original matrices. A reduction operation $red_{\mathtt{G},\mathtt{M}}(J)$ eliminates all rows and columns from a matrix J which do not correspond to elements in $\mathtt{G}$ and $\mathtt{M}$, respectively. The following holds for context composition: $\mathcal{K}_1 \diamond \mathcal{K}_2 = (\mathtt{G}_1, \mathtt{M}_1 \cup \mathtt{G}_2, I \sqcup nul_{\mathtt{G}_1,\mathtt{G}_2}) \circ (\mathtt{M}_1 \cup \mathtt{G}_2, \mathtt{M}_2, J \sqcup nul_{\mathtt{M}_1,\mathtt{M}_2})$

Table 4. Further context operations

$\mathcal{K}_1 \cup \mathcal{K}_2$	$:= \mathcal{K}_1 \sqcup \mathcal{K}_2$ if $\mathtt{G}_1 = \mathtt{G}_2, \mathtt{M}_1 = \mathtt{M}_2$
$\mathcal{K}_1 \circ \mathcal{K}_2$	$:= \mathcal{K}_1 \diamond \mathcal{K}_2$ if $\mathtt{M}_1 = \mathtt{G}_2$
$\frac{\mathcal{K}_1}{\mathcal{K}_2}$	$:= \mathcal{K}_1 \sqcup \mathcal{K}_2$ if $\mathtt{G}_1 \cap \mathtt{G}_2 = \emptyset$ and $\mathtt{M}_1 = \mathtt{M}_2$
$\mathcal{K}_1 \vert \mathcal{K}_2$	$:= \mathcal{K}_1 \sqcup \mathcal{K}_2$ if $\mathtt{G}_1 = \mathtt{G}_2$ and $\mathtt{M}_1 \cap \mathtt{M}_2 = \emptyset$

$dia_{\mathtt{G}}(\mathtt{S}_{\rightarrow})$	see definition 9
$dia_{\mathtt{M}}(\mathtt{S}_{\uparrow})$, $dia_{\mathtt{G}}(I_{\rightarrow})$, $dia_{\mathtt{M}}(I_{\uparrow})$	analogous to $dia_{\mathtt{G}}(\mathtt{S}_{\rightarrow})$
$\mathtt{set}_{\mathtt{G}}(I)$, $\mathtt{set}_{\mathtt{M}}(I)$	see definition 9, $\mathtt{set}_{\mathtt{G}}(I) = \mathtt{set}_{\mathtt{G}}(dia_{\mathtt{G}}(I_{\rightarrow}))$
$nul_{\mathtt{G},\mathtt{M}}$	$:= \overline{I_{\mathtt{G},\mathtt{M}} \cup \overline{I_{\mathtt{G},\mathtt{M}}}}$
$red_{\mathtt{G},\mathtt{M}}(J)$	$:= dia_{\mathtt{G}} \diamond J_{\mathtt{G}_1,\mathtt{M}_1} \diamond dia_{\mathtt{M}}$
$col_{\mathtt{G}}(\mathtt{S})$	$:= dia_{\mathtt{G}}(\mathtt{S}_{\rightarrow}) \circ one_{\mathtt{G},\{\mathtt{x}\}}$
$row_{\mathtt{M}}(\mathtt{S})$	$:= one_{\{\mathtt{x}\},\mathtt{M}} \circ dia_{\mathtt{M}}(\mathtt{S}_{\uparrow})$

Definition 9 A *context algebraic structure (CAS) based on* ACT is a three sorted algebra $(R_1, R_2, R_3, \sqcup, ^-, \diamond, ^d, dia(), \mathtt{set}(), (,,))$ where R_2 is a set of subsets of $\mathtt{ACT}$, R_3 is a set of Boolean matrices, R_1 is a set of formal contexts based on $\mathtt{ACT}$ and constructed using the partial function $(,,) : R_2^2 \times R_3 \rightarrow R_1$; $\sqcup, ^-, \diamond, ^d$ are according to definition 8; $\mathtt{set}_{\mathtt{G}}(I) := \{\mathtt{g} \in \mathtt{G} \mid \exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{g} I \mathtt{m}\}$; $\mathtt{set}_{\mathtt{M}}(I) := \{\mathtt{m} \in \mathtt{M} \mid \exists_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} I \mathtt{m}\}$; and $dia_{\mathtt{G}}(\mathtt{S}_{\rightarrow})$ is defined by $(i,j)_{dia_{\mathtt{G}}(\mathtt{S}_{\rightarrow})} = 1$ iff $i = j$ and for the ith element in $\mathtt{G}$: $\mathtt{g}(\mathtt{i}) \in \mathtt{S}$.

The algebra in definition 9 is not a RA because formal contexts have different nul elements and composition from the left and the right may require a different dia element. But presumably a homomorphism can be constructed that maps each context $(\mathtt{G}, \mathtt{M}, I)$ onto a pair $(sqr(\mathtt{G}, \mathtt{M}), I)$, that maps all diagonal matrices onto dia and all null matrices onto nul, and that maps the other operations accordingly resulting in a context-RA. (The details of this are left to future research.)

Table 5. Basic FCA operations in the named perspective

standard FCA	CAS
$\mathtt{g} I \mathtt{m}$	$\underline{g}^d \circ I \circ \underline{m}^d = (1)$
$\mathtt{g}' := \{\mathtt{m} \in \mathtt{M} \mid \mathtt{g} I \mathtt{m}\}$	$\underline{g}' = \underline{g}^+ := \underline{g}^d \circ I$
$\mathtt{H}' := \{\mathtt{m} \in \mathtt{M} \mid \forall_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \Longrightarrow \mathtt{g} I \mathtt{m}\}$	$H' := \overline{H^d \circ \overline{I}}$
$\mathtt{H}^+ := \{\mathtt{m} \in \mathtt{M} \mid \exists_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \text{ and } \mathtt{g} I \mathtt{m}\}$	$H^+ := H^d \circ I$

Lemma 3 In the named perspective the basic FCA operations can be represented as summarised in table 5.

The following conventions are used in table 5: for a context $(\mathtt{G}, \mathtt{M}, I)$; $\mathtt{H} \subseteq \mathtt{G}$; $\mathtt{N} \subseteq \mathtt{M}$; $\mathtt{g} \in \mathtt{G}$; $\mathtt{m} \in \mathtt{M}$; $H := col_{\mathtt{G}}(\mathtt{H})$; $\underline{g} := col_{\mathtt{G}}(\{\mathtt{g}\})$. As declared in footnote 3, the matrix names derived from elements of $\mathtt{ACT}$ are underlined (such as $\underline{g}$). In the first row, (1) is a 1×1 matrix with element 1. In the named perspective, sets are best represented as row or column matrices. Because $\mathtt{G}$ and $\mathtt{M}$ need not be disjoint, it can be ambiguous whether $\underline{g}$ is a row or column. In that case, the notations $\underline{g}_c := col_{\mathtt{G}}(\{\mathtt{g}\})$ and $\underline{g}_r := row_{\mathtt{M}}(\{\mathtt{g}\})$ can be used. In table 5, H' is a row matrix but H and H'' are column matrices. H'' is calculated dually to H' by composition with I from the left: $H'' = \overline{\overline{I} \circ \overline{H^d \circ \overline{I}}^d} = \overline{\overline{I} \circ \overline{\overline{I}^d \circ H}}$. The notations from the unnamed and named perspective are compatible with each other and can be used together.

4.4 Eight quantifiers

The use of negation and composition in the calculation of H' and H^+ raises the question as to whether other combinations of negation and composition are of interest. Table 6 summarises all 8 possible combinations of negation and composition for a context $(\mathtt{G}, \mathtt{M}, I)$ and a set $\mathtt{N} \subseteq \mathtt{G}$. The third column in that table provides a rough linguistic description, which should be taken with caution because words such as "only" are fairly ambiguous in natural languages. In many applications, these 8 quantifiers result in 8 different sets, which together describe the relationship of $\mathtt{N}$ and $\mathtt{G}$ in some detail. In the next section, an example of a lattice construction is provided that summarises all 8 quantifiers in one diagram. It should be noted that with respect to relational databases, it can be quite challenging to formulate these 8 quantifiers in SQL because the "ALL" quantifier (corresponding to the so-called relational division in RLA) is not a primitive operation in SQL. In fact to represent this "ALL" quantifier in SQL, two sub-select statements are required (cf. Priss (2005) for an example).

Table 6. Eight Quantifiers

$\mathtt{N}^+$	$I \circ N$	at least one, some
$\mathtt{G} \setminus \mathtt{N}^+$	$\overline{I \circ N}$	none
$\mathtt{N}'$	$\overline{\overline{I} \circ N}$	relates to all
$\mathtt{G} \setminus \mathtt{N}'$	$\overline{I} \circ N$	does not relate to all
$(\mathtt{M} \setminus \mathtt{N})^+$	$I \circ \overline{N}$	relates to those that are not only
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})^+)$	$\overline{I \circ \overline{N}}$	relates to those that are only
$(\mathtt{M} \setminus \mathtt{N})'$	$\overline{\overline{I} \circ \overline{N}}$	relates to all outwith
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})')$	$\overline{I} \circ \overline{N}$	does not relate to all outwith

4.5 Compositional schemata

To represent more complex data than just a single relation using RA some kind of canonic means for translating complex data into binary relations is needed.

Definition 10 A *compositional schema* consists of a set of 4 or 9 formal contexts, which are arranged in a tabular manner, (cf. figure 1) so that some of the contexts can be derived from adjacent contexts using composition.

The idea of compositional schemata is not new. There have been many papers on FCA which use such schemata explicitly or implicitly (eg. Ganter & Wille (1999), Priss (1998), Faid et al. (1997)). By identifying certain types of compositional schemata, their properties can be described in a general manner.

	A	B
A	1	2
C	3	4

	A	B	C
B	1	2	3
A	4	5	6
D	7	8	9

	A	B	C
B			L
A		J	$J \circ L$
D	I	$I \circ J$	$I \circ J \circ L$

Fig. 1. A compositional schema

The numbering of the four or nine cells as presented in the left hand side of figure 1 is used in the remainder of this paper. In the case of nine cells, the compositional schema is built from the formal contexts $K_J := (\mathtt{A}, \mathtt{B}, J)$; $K_I := (\mathtt{D}, \mathtt{A}, I)$ and $K_L := (\mathtt{B}, \mathtt{C}, L)$. Because K_I and K_J share the set $\mathtt{A}$, a context $K_{I \circ J} := (\mathtt{D}, \mathtt{B}, I \circ J)$ can be formed. Similarly, a context $K_{J \circ L}$ can be formed. Instead of the existence quantifier used in the construction of the matrices in cells 6, 8 and 9, any of the other seven quantifiers from table 6 can be used. A further context $K_{I \circ J \circ L}$ can be formed in cell 9 to complete the schema. It should be noted that while $K_{J \circ L}$ and $K_{I \circ J}$ are formed by composing the context to the left with the one above, $K_{I \circ J \circ L}$ is formed by composing the context to the left with the context two steps above (or the context two steps to the left with the one above). An exception is if J is a reflexive, transitive relation, in which case $J \circ J = J$ and $I \circ J \circ L = I \circ J \circ J \circ L$. Depending on the application, cells 1, 2, 4 can be filled with nul or dia or something else. In many cases, it may not be necessary to calculate a lattice for the context consisting of all 9 cells, but instead only for cells 5, 6, 8, 9 or just for individual cells. For identifying where objects and attributes are located in the schema, row and column matrices representing the sets $\mathtt{A}$, $\mathtt{B}$, and so on can be used. For example, an element $\mathtt{g}$ is an object in K_J if $\underline{g}_c \subseteq A_c$.

Figure 2 provides two examples of compositional schemata. The first example can be constructed for any concept lattice. This example shows a formal context $(\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}, \{1, 2, 3, 4\}, I)$ where I is the matrix in cell 4 of the schema. After computing the set of concepts of this lattice, ($\{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$ without the top and bottom concept), cell 1 is filled with the conceptual hierarchy I_{sub}; cell 2 is filled with the intension relation between attributes and concepts (here called I_{attr}); cell 3 is filled with the extension relation between objects and concepts (called I_{inst}). Because I_{sub} is reflexive and transitive: $I = I_{inst} \circ I_{sub} \circ I_{attr} = I_{inst} \circ I_{sub} \circ I_{sub} \circ I_{attr}$. In this case also $I_{inst} = I_{inst} \circ I_{sub}$ because $\exists_{\mathtt{c}_1} : \mathtt{g} I_{inst} \mathtt{c}_1,\ \mathtt{c}_1 I_{sub} \mathtt{c}_2 \iff \mathtt{g} I_{inst} \mathtt{c}_2$ and the same for I_{attr}.

The bottom half of figure 2 shows a lattice that visualises all 8 quantifiers from table 6. The compositional schema is constructed by inserting dia into cell 1, I into cell 3, the $\in$ relation into cell 2, and $I \circ N$ into cell 4. The lattice diagram shows that $I \circ N$ and $\overline{\overline{I} \circ N}$ are the intensions of the join and meet of the elements in N. $I \circ \overline{N}$ and $\overline{\overline{I} \circ \overline{N}}$ are the intensions of the join and meet of $M \setminus N$ which is $\overline{N}$. The other 4 quantifiers need

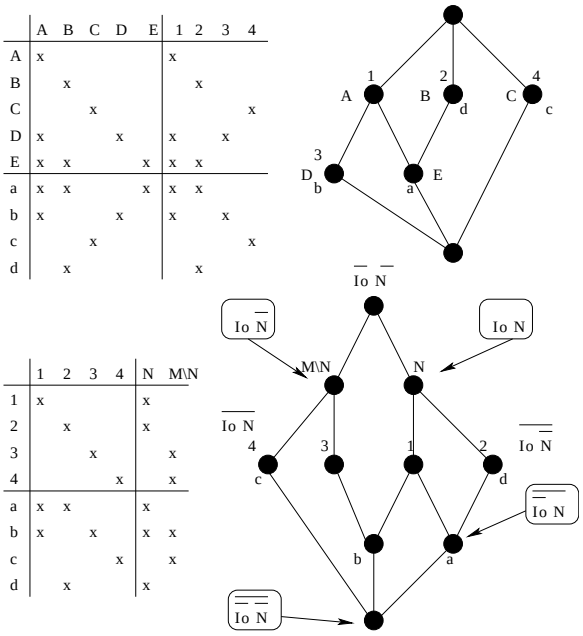

Fig. 2. Two examples of compositional schemata

not correspond to single concepts in the lattice but are the set-complements of the first 4 quantifiers. This lattice has the original lattice $(\mathtt{G}, \mathtt{M}, I)$ as a sublattice. If this lattice was produced as an answer to a query about elements in N, it would answer many questions simultaneously: whether elements are at least in N, not in N, just in N, and so on.

5 Relational Schemata and Fork Algebra

5.1 Relational Schemata

This section covers schemata that represent the table structure of a relational database but without the actual values that are stored in the database and without showing which tables correspond to what is called "entities" and what is called "relations" in relational databases. More complex schemata with values and relations are covered in the next section. Relational schemata are relevant not just for relational databases but also for object-relational databases. There are some differences between the implementations (and thus the underlying formalisations) of object-relational databases among different vendors. For this paper, object-relational databases are considered to have a subtype relation among tables, i.e., one table can be defined to be a subtype of another table. This subtype relation is declared to be reflexive, acyclic and transitive. A subtype table inherits all columns (attributes) from its supertypes and the instances (rows) of a subtype are also assigned to its supertypes after deleting non-applicable columns. With

this definition, a relational database is an object-relational database where the subtype relation is the identity (each table is only subtype of itself).

Definition 11 A *relational schema (of an object-relational database) based on* ACT is a CAS using a compositional schema according to figure 3 where Tbls is a set of table names, Inst a set of instances and Attr a set of column names with $\texttt{Inst} \subseteq \texttt{ACT}$; $\texttt{Tbls}, \texttt{Attr} \subseteq \texttt{N}$, and the sets are pairwise disjoint and linearly ordered according to ACT. The subschema consisting of the cells 5, 6, 8, 9 is denoted by $\mathcal{DB}$.

	Tbls	Tbls	Attr
Tbls			I_{attr}
Tbls		I_{sub}	$I_{sub} \circ I_{attr}$
Inst	I_{inst}	$I_{inst} \circ I_{sub}$	$I_{isba} := I_{inst} \circ I_{sub} \circ I_{attr}$

Fig. 3. The basic relational schema for an object-relational database

It is normally assumed that I_{attr} has no empty columns and that I_{inst} has no empty rows because having instances or attributes which are not in relationship to anything else is strange. The relational schema $\mathcal{DB}$ is basically the same as the first example of figure 2 because I_{sub} is reflexive and transitive and thus, for example, $I_{inst} \circ I_{sub} = I_{inst} \circ I_{sub} \circ I_{sub}$. In the line diagram of the lattice of $\mathcal{DB}$ the name of a table is attached to a node both as an object and as an attribute or in other words:

Theorem 1 For each table $\texttt{t} \in \texttt{Tbls}$ there exists a formal concept $c(\texttt{t})$ in the concept lattice of $\mathcal{DB}$ which has t in its contingent extent and in its contingent intent. If $\texttt{t}_1 \neq \texttt{t}_2$ then $c(\texttt{t}_1) \neq c(\texttt{t}_2)$.

Proof: as before $\texttt{t}_\texttt{r}$ denote a table among the attributes and $\texttt{t}_\texttt{c}$ the same table among the objects. $\texttt{t}'_r = \{\texttt{s}_c \in \texttt{Tbls} \mid \texttt{s}_c I_{sub} \texttt{t}_r\} \cup \{\texttt{i} \in \texttt{Inst} \mid \texttt{i}(I_{inst} \circ I_{sub})\texttt{t}_r\}$ and $\texttt{t}''_c = \{\texttt{s}_r \in \texttt{Tbls} \mid \forall_{\texttt{y} \in \texttt{t}'_c} : \texttt{s}_r(I_{sub} | I_{sub} \circ I_{attr})\texttt{y}\} \cup \{\texttt{i} \in \texttt{Inst} \mid \forall_{\texttt{y} \in \texttt{t}'_c} : \texttt{i}(I_{inst} \circ I_{sub} | I_{isba})\texttt{y}\}$. Because of $\texttt{s} I_{sub} \texttt{t}_r \iff \forall_{\texttt{y} \in \texttt{t}'_c} : \texttt{s}(I_{sub} | I_{sub} \circ I_{attr})\texttt{y}$ and $\texttt{i}(I_{inst} \circ I_{sub})\texttt{t}_r \iff \forall_{\texttt{y} \in \texttt{t}'_c} : \texttt{s}(I_{inst} \circ I_{sub} | I_{isba})\texttt{y}$ it follows that $\underline{t}'_r = \underline{t}''_c$. Same for $\underline{t}'_c = \underline{t}''_r$. This implies that $(\texttt{t}''_c, \texttt{t}'_c)$ and $(\texttt{t}'_r, \texttt{t}''_r)$ describe the same concept which has t in its contingent. Because the subtype relation I_{sub} is assumed to be acyclic (i.e., a table cannot be subtype of a second table which is itself a subtype of the first table), there is a different formal concept in $\mathcal{DB}$ for each different table.

The concept $\texttt{c}(\texttt{t})$ has all attributes of t in its intension and all instances of t in its extension. Thus the concept lattice of $\mathcal{DB}$ summarises important information about the tables of an object-relational database. Another feature of $\mathcal{DB}$ is that different types of inheritance can be defined and analysed (cf. Priss (2005)). The formal context C_t for a table t can be derived as $C_t = (\texttt{set}_{\texttt{Inst}}(\underline{t_c}'), \texttt{set}_{\texttt{Attr}}(\underline{t_r}'), red_{\texttt{G}_t, \texttt{M}_t}(I_{inst} \circ dia_{\texttt{Tbls}}(\underline{t}) \circ I_{attr}))$.

5.2 Fork Algebraic definitions

Tarski showed that RA is equivalent to first order logic with three variables (cf. Van den Bussche (2001)). An indication for why three variables are sufficient is given by Van den Bussche's example: $\{(x, y) \mid \exists_z (\exists_y (\exists_z (R(x, z) \wedge R(z, y)) \wedge R(y, z)) \wedge R(z, y))\}$. Tarski further showed that what is missing from RA is a form of "pairing", i.e., a means

for combining two elements into a pair which then itself behaves like a primitive element. This pairing is required to build ternary relations. Different methods for adding a "pairing axiom" to RA have been suggested (eg. Jain, Mendhekar & Van Gucht (1995)). The approach which seems to be most widely used and which indeed has expressive power equivalent to first order logic is called "fork algebra" (Frias et al., 2004). It was developed in the area of programming language semantics for the purpose of dealing with non-deterministic algorithms. To our knowledge, applications in the area of databases as we are suggesting in this paper have not been discussed before.

The following two definitions are adapted from (Frias et al., 2004). For the purposes of this paper the usual operation (∇), is replaced by its relational dual denoted by $\triangle$.

Definition 12 A *fork algebra* is an algebra $(R, +, \cdot, ', 0, 1, ;, ^\smile, e, \triangle)$ so that $(R, +, \cdot, ', 0, 1, ;, ^\smile, e)$ is a relation algebra and for all $r, s, t, u \in R$:

F1 $r \triangle s = ((e \triangle 1); r) \cdot ((1 \triangle e); s)$

F2 $(r^d \triangle s^d)^d; (t \triangle u) = (r; t) \cdot (s; u)$

F3 $(e \triangle 1)^d \triangle (1 \triangle e)^d \leq e$

Definition 13 A *pre-proper fork algebra* (FRA) is a two sorted algebra $(R, \mathtt{U}, \cup, \cap, ^-, nul, one, \circ, ^d, dia, \triangle, \mathtt{frk}())$ where $(R, \cup, \cap, ^-, nul, one, \circ, ^d, dia)$ is a RA on a set $\mathtt{U}$; a binary function $\mathtt{frk} : \mathtt{U} \times \mathtt{U} \to \mathtt{U}$ is injective on the restriction of its domain to one; the operation $\triangle$ is defined as $I \triangle J := \{(\mathtt{frk}(\mathtt{x}, \mathtt{y}), \mathtt{z}) \mid (\mathtt{x}, \mathtt{z}) \in I; (\mathtt{y}, \mathtt{z}) \in J\}$ and R is closed under $\triangle$.

Proper fork algebras are defined somewhat more abstractly than pre-proper ones, but they are not required for this paper. Unless $one = dia$, the `frk` operation in definition 13 requires an infinite set of elements because of the injectivity. It should be noted that `frk` is not associative, thus usually $\mathtt{frk}(\mathtt{frk}(\mathtt{x}, \mathtt{y}), \mathtt{z}) \neq \mathtt{frk}(\mathtt{x}, \mathtt{frk}(\mathtt{y}, \mathtt{z}))$. With respect to active domains, the `frk` operation in this paper has the purpose of assigning unique identifiers.

Definition 14 A *context-FRA based on $\mathcal{A}$* is a context-RA based on $\mathcal{A}$ with a FRA on `ACT` which fulfills the following: with $\mathtt{frk} : \mathtt{U}_1 \times \mathtt{U}_2 \to \mathtt{U}_3$: if $\mathtt{U}_1 = \mathtt{U}_2 = \mathtt{ACT} \Rightarrow \mathtt{U}_3 = \mathtt{ODD} \cup \mathtt{EVN}$ and if $\mathtt{U}_1 = \mathtt{ODD}$ or $\mathtt{U}_2 = \mathtt{ODD} \Rightarrow \mathtt{U}_3 = \mathtt{ODD}$. For $\mathtt{x}, \mathtt{y} \in \mathtt{EVN}$: $\mathtt{frk}(\mathtt{x}, \mathtt{y}) \neq \mathtt{x}$ and $\mathtt{frk}(\mathtt{x}, \mathtt{y}) \neq \mathtt{y}$. The following restrictions to `ACT` are defined: $I \triangle|_{\mathtt{EVN}} J := \{(\mathtt{frk}(\mathtt{x}, \mathtt{y}), \mathtt{z}) \mid (\mathtt{x}, \mathtt{z}) \in I; (\mathtt{y}, \mathtt{z}) \in J; \mathtt{frk}(\mathtt{x}, \mathtt{y}) \in \mathtt{EVN}\}$ and $lft := dia_{\mathcal{A}} \triangle|_{\mathtt{EVN}} one_{\mathcal{A}}$ and $rgt := one_{\mathcal{A}} \triangle|_{\mathtt{EVN}} dia_{\mathcal{A}}$ and $prt := rgt \sqcup lft \sqcup (rgt \diamond lft) \sqcup (lft \diamond rgt) \sqcup (rgt \diamond rgt) \sqcup (lft \diamond lft)...$ and $end := dia(prt_\uparrow) \cap dia(prt_\rightarrow)$.

It can be shown that F1 and F3 (but not F2) from definition 12 still hold for $\triangle|_{\mathtt{EVN}}$. lft and rgt are projections because $lft = \{(\mathtt{frk}(\mathtt{x}, \mathtt{y}), \mathtt{x}) | \, \mathtt{frk}(\mathtt{x}, \mathtt{y}) \in \mathtt{EVN}\}$. According to F1, all of the information about $\triangle|_{\mathtt{EVN}}$ is contained in lft and rgt. The matrices lft and rgt are fixed at any point in time according to `EVN`. Calculations with $\triangle|_{\mathtt{EVN}}$ are thus reduced to look-ups in lft and rgt together with ordinary RA operations. It should be noted that to calculate the parts, prt requires some sort of transitive closure (thus is not strictly an RA operation). Frias et al. (2004) do not discuss the need for transitive closure, but it is not known to us whether they do not require it or whether they have overlooked the problem.

5.3 Relational Schemata using Fork Algebra

Using the fork algebraic extensions from the previous section, it is now possible to define a complete relational schema for an object-relational database that contains both

simple and composite tables with all their values. Simple tables are traditionally called "entity tables". They collect instances, such as "employee" or "project". Instances (or rows) in such tables are usually identified by a single key, which is a column of the table and contains unique values, such as "employee number" or "project number". Composite tables are traditionally called "relations", which are built using the keys from simple tables as "foreign keys". For example, a relation "work" can be built from tables "employee" and "project" using the keys "employee number" and "project number". Such a table represents a database relation between employees and projects.

Definition 15 A *complete relational schema based on* ACT is a relational schema based on ACT with a context-FRA based on $\mathcal{A}$ and with sets: $\mathtt{Keys} \subseteq \mathtt{Attr}$; $\mathtt{Nkey} := \mathtt{Attr} \setminus \mathtt{Keys}$; $\mathtt{Simp} \subseteq \mathtt{Inst}$ with $\mathtt{Simp} := \{\mathtt{s} \in \mathtt{Inst} | \neg\exists_{\mathtt{y}} : \mathtt{s}\, prt\, \mathtt{y}\} \cup \{\mathtt{s} \in \mathtt{ACT} \,|\, \exists_{\mathtt{x}} : \mathtt{x}\, end\, \mathtt{s}\}$ and $\mathtt{Comp} := \mathtt{Inst} \setminus \mathtt{Simp}$; so that simple instances have at most one key: for $\mathtt{s} \in \mathtt{Simp}, \mathtt{k}_1, \mathtt{k}_2 \in \mathtt{Keys}$: $\mathtt{s} I_{isba} \mathtt{k}_1, \mathtt{s} I_{isba} \mathtt{k}_2 \Rightarrow \mathtt{k}_1 = \mathtt{k}_2$ and the keys of composite instances correspond exactly to their fork algebraic end parts: for $\mathtt{c} \in \mathtt{Comp}$: $\exists_{\mathtt{s}} : \mathtt{c}\, end\, \mathtt{s} \Longleftrightarrow \exists_{\mathtt{k} \in \mathtt{Keys}} : \mathtt{c}\, end \diamond I_{isba}\, \mathtt{k}$; for $\mathtt{c} \in \mathtt{Comp}, \mathtt{k} \in \mathtt{Keys}$: $\mathtt{c}\, end \diamond I_{isba}\, \mathtt{k} \Longleftrightarrow \mathtt{c} I_{\mathtt{isba}} \mathtt{k}$ and $|\mathtt{set}((\underline{c} \diamond end)_\uparrow))| = |\mathtt{set}_{\mathtt{Keys}}((\underline{c} \diamond I_{isba})_\uparrow)|$.

Definition 15 does not allow for the same attribute to be used more than once as a foreign key. This is only a problem if these attributes are in a many-to-many relation because otherwise the relation does not require a separate table. But even then it is possible to generate a generic key attribute using identifiers and treating the other attributes as non-key attributes.

Remark 1 Definition 15 translates the database notion that instances are uniquely identified by keys into a fork algebraic part-whole relationship! This is significant because in RLA, keys form just another set and instance pairs are not structurally different from value pairs. But in the fork algebraic formalisation, the special nature of keys is structurally represented.

The conditions in definition 15 can also be expressed relationally: simple instances do not have parts: $dia(\mathtt{Simp}) \diamond prt = nul$. End parts are simple: $dia(\mathtt{Simp}) \supseteq dia(end_\uparrow)$ or $end = prt \diamond dia(\mathtt{Simp})$. Simple instances have exactly one key: $red_{\mathtt{Simp},\mathtt{Keys}}(I_{isba})$ is a permutation matrix[4]. Keys of composite instances correspond to the fork algebraic end parts of the instances: for each $\mathtt{c} \in \mathtt{Comp}$: $dia_{\mathtt{set}}((\underline{c} \diamond end)_\uparrow) \diamond I_{isba} \diamond dia_{\mathtt{Keys}}((\underline{c} \diamond I_{isba})_\uparrow)$ is a permutation matrix.

So far, instances can be constructed and attributes can be assigned to instances – but only in a binary manner showing which instance has which attribute but not which value belongs to an instance with respect to an attribute. In the following definition, Vals stands for instance-value pairs. Attribute values are often drawn from potentially infinite domains (such as the set of real numbers). This is why attribute values do not usually correspond to keys. According to Definition 15 simple instances do not need to have a key. Simple instances without keys are attribute values. But not all attribute values need to be listed as instances in a complete relational schema.

Definition 16 A *value assignment context* for a complete relational schema based on ACT is a formal context $(\mathtt{Vals}, \mathtt{Nkey}, I_{vals})$ where $\mathtt{Vals} \subseteq \mathtt{ACT}$ with $dia(\mathtt{Vals}) \subseteq dia(lft_\rightarrow)$ so that $((dia(\mathtt{Vals}) \diamond lft)_\uparrow) \subseteq dia(\mathtt{Inst})$ and I_{vals} is a binary matrix so

[4] A binary relation I represents an exact correspondence, if it contains exactly one 1 in each row and column (i.e., $I \circ I^d = dia$). Such a matrix is called a permutation matrix.

that for each attribute a: the matrix $dia((I_{vals} \diamond \underline{a})_{\rightarrow}) \diamond lft = dia(\mathtt{set}(\underline{a}')) \diamond lft$ has at most 1 cross per column.

The conditions in definition 16 ensure that elements of Vals are pairs where the left element is an instance. The instance-value relation for each attribute can be retrieved via $lft^d \diamond dia(\mathtt{set}(\underline{a}')) \diamond rgt$. The last condition in definition 16 ensures that each instance has exactly one value for each attribute. In relational database terms this means that the tables are in first normalform. Definition 16 does not only require this for each single table, but instead across the whole database. If an instance has a value for an attribute, then it has the same value in all tables in which this instance and this attribute occur. This means that a multiple inheritance anomaly (Priss, 2005) is avoided. From an implementation viewpoint, this can always be achieved by renaming attributes if these attributes have table-specific values.

The construction in definition 16 is in principle similar to the treatment of "many-valued contexts" in traditional FCA (Ganter & Wille, 1999) and to Wille's (2002) power context families. The difference is, however, that in definition 16 a single formal context is used for all attributes of all database tables of an application. The information about which instances and values belong to which database tables is coded into the fork algebraic part-whole structure of the elements in Vals. The fork algebraic construction also uses a more restrictive set of operations. Instead of the use of a Cartesian Product, pairs of instances or of instances and values are added to lft and rgt on an as-needed basis. Another restriction (but not limitation) is that n-ary relations are built stepwise from binary relations.

6 Conclusion

This paper describes a formalisation of object-relational databases using RA and fork algebra. An advantage for this approach is that in contrast to traditional RLA, the mathematisation is mainly based on binary relations and thus closer to FCA, which provides easy access to visualisations in form of FCA line diagrams. Compared to RLA, the basis of the algebraic operations that are required is quite similar. Both RA and RLA use union and complement. The RLA operations of projection and selection are achieved in RA by using composition, dual and a selection via composition with $dia()$. The RLA operation of cross (or Cartesian) product corresponds to fork algebraic constructions together with RA operations that allow to convert between sets and matrices. The similarities and differences between the two approaches provide insights into the structure of relational databases. It is hoped that in the future an implementation can be developed that explores practical applications of the RA/fork algebraic structures.

References

1. Abiteboul, Serge; Hull, Richard; Vianu, Victor (1995). *Foundations of databases*. Addison Wesley.
2. Atkinson, M.; Bancilhon, F.; DeWitt, D.; Dittrich, K.; Maier, D.; Zdonik, S. (1989). *The Object-Oriented Database System Manifesto.* In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan. p, 223-240.

3. Brink, C.; Britz, K.; Schmidt, R. (1992). *Peirce Algebras.* Max-Planck-Institut für Informatik. MPI-I-92-229.
4. Codd, E. (1970). *A relational model for large shared data banks.* Communications of the ACM, 13:6.
5. Faid, M.; Missaoui, R.; Godin, R. (1997). *Mining Complex Structures Using Context Concatenation in Formal Concept Analysis.* In: Mineau, Guy; Fall, Andrew (eds.), Proceedings of the Second International KRUSE Symposium. p. 45-59.
6. Frias, Marcelo; Veloso, Paulo; Baum, Gabriel (2004). *Fork Algebras: Past, Present and Future*. Journal on Relational Methods in Computer Science, 1, p. 181-216.
7. Ganter, B.; Wille, R. (1999). *Formal Concept Analysis.* Mathematical Foundations. Berlin-Heidelberg-New York: Springer, Berlin-Heidelberg.
8. Hansen, Gary; Hansen, James (1988). *Human Performance in Relational Algebra, Tuple Calculus, and Domain Calculus.*, International Journal of Man-Machine Studies, 29, 503-516.
9. Hereth, J. (2002). *Relational Scaling and Databases.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag. p. 62-76.
10. Imielinski, T.; Lipski, W. (1984). *The relational model of data and cylindric algebras.* Journal of Computer and Systems Sciences, 28, p. 80-102.
11. Jain, M.; Mendhekar, A.; Van Gucht, D. (1995). *A Uniform Data Model for Relational Data and Meta-Data Query Processing.* International Conference on Management of Data.
12. Kim, K. H. (1982). *Boolean Matrix Theory and Applications.* Marcel Dekker Inc.
13. Lyndon, R. (1950). *The representation of relational algebras.* Ann. of Math., 2, 51, p. 707-729.
14. Maddux R. (1996). *Relation-algebraic semantics.* Theoretical Computer Science, 160, p. 1-85.
15. Pratt, V.R. (1992). *Origins of the Calculus of Binary Relations.* Proc. IEEE Symp. on Logic in Computer Science, p. 248-254.
16. Pratt, V.R. (1993). *The Second Calculus of Binary Relations.* Proc. 18th International Symposium on Mathematical Foundations of Computer Science, Gdansk, Poland, Springer-Verlag, p. 142-155.
17. Priss, U. (1998). *Relational Concept Analysis: Semantic Structures in Dictionaries and Lexical Databases.* (PhD Thesis) Verlag Shaker, Aachen 1998.
18. Priss, U.; Old, L. J. (2004). *Modelling Lexical Databases with Formal Concept Analysis.* Journal of Universal Computer Science, Vol 10, 8, 2004, p. 967-984.
19. Priss, U. (2005). *Establishing connections between Formal Concept Analysis and Relational Databases.* In: Dau; Mugnier; Stumme (eds.), Common Semantics for Sharing Knowledge: Contributions to ICCS 2005, p. 132-145.
20. Tarski, A. (1941). *On the calculus of relations.* Journal of Symbolic Logic, 6, p. 73-89.
21. Tarski, A. (1955). *Contributions to the theory of models.* Indag. Math, 17, p. 56-64.
22. Van den Bussche, Jan (2001). *Applications of Alfred Tarski's Ideas in Database Theory.* Proceedings of the 15th International Workshop on Computer Science Logic. LNCS 2142, p. 20-37.
23. Wagner, Stefan (2003). *Transitive closure in relational database systems.* On-line available at http://www.stefan-wagner.info/cs/trans_clos.php.
24. Wille, Rudolf (2002). *Existential Concept Graphs of Power Context Families.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag, p. 382-395.