Brock University

Department of Computer Science

# Cylindric algebras and relational databases

Ivo Düntsch

Technical Report # CS-09-05
April 2009

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

# Cylindric algebras and relational databases

Ivo Düntsch*
Department of Computer Science
Brock University
St. Catharines, ON, Canada
duentsch@brocku.ca

April 14, 2009

**Abstract**

An interpretation of the relational data model and various dependencies is given within the framework of cylindric algebras. It turns out that there is a natural translation of relational databases into cylindric set lattices, and that Codd's relational operators have a simple interpretation in these structures. Consequently, various database constraints can be expressed in the equational language, and recent results in equational logic have been able to shed some light on the expressiveness and axiomatizability of these constraints.

## 1 Introduction

The success of the relational data model introduced by Codd [4] can – at least partly – be attributed to its clarity and succinctness. The semantics of the model are constraints among and within the base relations which, as it turned out, could be formulated in various fragments of first order logic. Therefore, a natural approach seems to be the general framework of cylindric structures. The two main examples of these are concrete algebras of $n$–ary relations on the one hand, and algebras of first order sentences on the other. It is interesting to note that throughout most of the research on

*The author gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada.

the theory of relational databases, this dichotomy can be observed, albeit in a different terminology. Codd's relational algebra and domain- or tuple calculus are well known examples for this phenomenon, and, keeping in mind the connection between cylindric algebras and first order logic, it comes as no surprise that the resulting query languages are of the same expressive power. The extended relations of [24] are, loosely speaking, nothing else than an embedding of some finite cylindric algebra into an infinite algebra of formulae, and a transformation from finite dimensional relations to formulae (and infinite dimensional cylindric algebras) is implicitly present in [7].

The plan for the current chapter is as follows: After a brief introduction to Codd's data model, I will present an embedding of relational data tables into cylindric structures similar to the first such construction by Imilienski and Lipski [17]. The rest of the chapter will be taken up by the discussion of various forms of data dependencies in algebraic setting.

I will use the definitions and notation of [13, 14].

## 2 The relational data model

The two components of Codd's original concept of a relational database are objects and operators [4]:

1. Suppose that $\alpha$ is a countable cardinal, and $\{D_n : n < \alpha\}$ is a collection of nonempty sets which are called *domains*. Intuitively, $\alpha$ is a set of attribute names, and a domain $D_n$ consists of those values which the $n$-th attribute of a data table may take. Even though an instance of a database is finite, the domains are usually allowed to be infinite so as to have enough choices for a field entry. If $\emptyset \neq X \subseteq \alpha$, a subset $r$ of $\prod_{i\in X} D_i$ is called a *data table over the schema* $X$, written as $\langle X, r\rangle$; the largest data table with schema $X$ is denoted by $\mathbf{1}_X$, i.e. $\mathbf{1}_X = \prod_{n\in X} D_n$. Note that data tables are (heterogeneous) relations. The elements of a data table are called *tuples* or *records*; observe that tuples are choice functions with domain $X$. If $r$ is a data table over $X$, we set $a(r) = X$, and call it the *schema of* $r$. A *database* is a structure $\mathcal{D} = \langle D_n, r_j\rangle_{n<\alpha, j\in J}$ such that, for every $j \in J$, $r_j$ is a data table with schema $a(r_j)$.

2. Besides the Boolean set operators $\cup$ and $\setminus$ which are defined on data tables with the same schema $X$, there are four operations specific to databases:

(a) *Join*: The *(natural) join* $\bowtie$ of two data tables $r$ and $s$ is defined as

$$r \bowtie s = \{f \in \mathbf{1}_{a(r)\cup a(s)} : f \restriction a(r) \in r, f \restriction a(s) \in s\}.$$

Observe that $a(r \bowtie s) = a(r) \cup a(s)$, and that the join operator glues r and s along their common attributes.

(b) *Project*: If $Y \subseteq X$, then the *projection operator* is defined by

$$\pi_Y(r) = \begin{cases} \{f \restriction Y : f \in r\}, & \text{if } Y \neq \emptyset, \\ \mathbf{1}_{a(r)}, & \text{otherwise.} \end{cases}$$

So, $\pi_Y$ picks all columns from $r$ whose attribute name is in $Y$.

(c) *Rename*: Suppose that $i \in a(r)$, $j \in \alpha \setminus a(r)$, and $D_i = D_j$. We need an operation which changes the name of column i to that of column j: Set $Y = (a(r) \setminus \{i\}) \cup \{j\}$, and define

$$\delta_{i \leftarrow j}(r) = \{f \in \mathbf{1}_Y : f \restriction (a(r) \setminus \{i\}) = g \restriction (a(r) \setminus \{i\}), \\ f(j) = g(i) \text{ for some } g \in r\}.$$

(d) *Select*: Select is a unary operator on relations, or, more precisely, a class of operators. For each attribute $n \in a(r)$ and each $d \in D_n$ let

$$\sigma_{n=d}(r) = \{f \in r : f(n) = d\}.$$

In other words, $\sigma_{n=d}(r)$ selects from $r$ all those tuples which have entry $d$ in column $m$.

# 3 An embedding

Suppose that $\mathcal{D} = \langle D_n, r_j \rangle_{n<\alpha, j\in J}$ is a database where $\alpha > 2$ and $\mathcal{R}$ the set of all its data tables. It is our aim to embed $\mathcal{D}$ into a $\mathsf{Cs}_\alpha$ over an appropriate base set, and to translate the operations among data tables into $\mathsf{Cs}_\alpha$ operations. To avoid the messiness of typed structures – and since we are not concerned with computational efficiency –, we shall assume throughout this section that the domains are all equal, say, $D_i = D$ . This may entail problems with respect to the selection operators when comparison of domain elements is involved. However, these can, in theory, be resolved, by

restricting the comparisons on $D$ to appropriate elements, so that there is no restriction of generality.

One difficulty we encounter when considering a data table $r$ is that $a(r)$ can be a proper subset of $\alpha$. However, in the translation we have in mind, we want to consider all occurring relations as $\alpha$-ary; in other words, data tables need to be interpreted as $\alpha$–ary relations without losing the significant columns $a(R)$. The most natural way to do this is to follow the philosophy that "all information is no information", and introduce dummy columns into $r$, see e.g. [20]. For example, if $a(r) = \alpha, X = \alpha \setminus \{0\}$, and $r = D \times \pi_X(r)$ then $\pi_X(r)$ carries the same information as $r$.

This method, however, has the drawback that it works only if $D \times r \notin \mathcal{R}$ for $r \in \mathcal{R}$. For example, let $D = \{a, b\}, \alpha = \mathbf{3}$, $r_0 = D \times \{a\} \times \{b\}$, and $r_1 = \langle a, b\rangle$ with. $\alpha(r_1) = \{1, 2\}$. The dummy embedding of $r_1$ into ${}^{\mathbf{3}}D$ is now equal to $r_0$ which is not what we want. If $\alpha = \omega$ and $a(r) < \omega$ for all $r \in \mathcal{R}$, then this problem does not occur. However, this does not seem a satisfactory solution, since it unnecessarily restricts our choice of domains. It is proposed in [17] to flag the irrelevant columns by an extra symbol; however, there it is not clear how the cylindric operations act on these extended relations. In the embedding below we shall also make use of a flag to identify irrelevant columns, though, in a different way.

Let $U = D \cup \{u\}$, where $u \notin D$, and let $V$ be the set of all $\alpha$–termed sequences over $U$. Define a mapping $e : \mathcal{R} \to 2^V$ by

$$e(r) = \{f \in V : f \restriction a(r) \in r\}.$$

**Theorem 3.1.** *Suppose that $\mathcal{D} = \langle D_n, r_j\rangle_{n<\alpha, j\in J}$ is a database such that $D_i = D$ for all $i < \alpha$, and let $\mathcal{R}$ be a collection of data tables over $\mathcal{D}$. Furthermore, $U$ and $e$ are defined as above. Then, $e$ is injective, and for all $r, s \in \mathcal{R}$, $Y \subseteq \alpha$,*

$$e(r \cup y) = e(r) \cup e(s) \qquad \text{if } a(r) = a(s). \tag{1}$$

$$e(r \setminus s) = e(r) \setminus e(s) \qquad \text{if } a(r) = a(s). \tag{2}$$

$$e(\pi_Y(r)) = c_{\alpha\setminus Y} e(r). \tag{3}$$

*Here, $c_{\alpha\setminus Y}$ is a generalized cylindrification, see [13], Ch. 1.7.*

$$e(r \bowtie s) = e(r) \cap e(s). \tag{4}$$

$$e(\delta_{i\leftarrow j}(r)) = s^i_j(e(r)), \tag{5}$$

*where $s^i_j$ denotes the "j for i substitution" of [13], Ch. 1.5.*

*Proof.* Let $r, s \in \mathcal{R}$ and $r \neq s$. If $a(r) \neq a(s)$, say, $n \in a(r) \setminus a(s)$, then there is some $f \in e(s)$ such that $f(n) = u$. On the other hand, $n \in a(r)$ implies that $u \neq f(n)$ for all $f \in e(r)$, and thus, $e(r) \neq e(s)$. If $a(r) = a(s)$ say, $f \in r \setminus s$. If $a(r) = a(s)$, then $r \neq s$ implies that w.l.o.g. there is some $f \in r \setminus s \neq \emptyset$. Thus, for all $g \in s$ there is some $n \in a(r) = a(s)$ with $f(n) \neq g(n)$. Hence, $e(r) \neq e(s)$.

The proofs of (1) – (5) are analogous to the corresponding properties of [17] and can safely be omitted. □

Recall that for any $r \in V$, *the dimension set of* $r$ is $\Delta(r) = \{n < \alpha : c_n(r) \neq r\}$ ([13], Ch. 1.6). The definition of $e$ shows that

$$n \in a(r) \Longleftrightarrow (\forall f \in V)[f \in e(r) \Rightarrow f(n) \neq u], \tag{6}$$

and

$$\begin{aligned} n \in \Delta(e(r)) &\Longleftrightarrow c_n(e(r)) \neq e(r), \\ &\Longleftrightarrow (\exists f \in V)[f \notin e(r) \text{ and } (\exists g \in e(r)(f(m) = g(m), m \neq n)], \\ &\Longleftrightarrow (\exists f \in V)[f(n) = u \text{ and } (\exists g \in e(r)(f(m) = g(m), m \neq n)], \\ &\Longleftrightarrow n \in a(r). \end{aligned}$$

By flagging irrelevant columns by a new element, we have, in a way, introduced a new constant, the algebraic version of which are regular thin elements ([14], p. 60): For each $n < \alpha$ let $v_n^u = \{f \in V : f(n) = u\}$. Then, each $e(r)$ satisfies

$$c_n(e(r)) \neq r \Rightarrow e(r) \cap v_n^u = \emptyset. \tag{7}$$

Observe that $\Leftarrow$ is always true for any nonzero $e(r)$. Generally, given a set $U$ and some $u \in U$, a set $r$ of $\alpha$ – termed sequences over $U$ is called a *db –relation (with respect to u)*, if $c_n(r) \neq r$ implies $r \cap v_n^u = \emptyset$ for all $n < \alpha$. We now have

**Theorem 3.2.** *1. Suppose that $\mathcal{D} = \langle D_n, r_j \rangle_{n<\alpha, j \in J}$ is a database such that $D_i = D$ for all $i < n$, and let $\mathcal{R}$ be a collection of data tables over $\mathcal{D}$. Furthermore, $U$ and $e$ are defined as above. Then, each $e(r)$ is a db–relation.*

*2. Conversely, if $r \subseteq {}^{\alpha}U$ is a db–relation with respect to some $u \in U$, then $r' = r \restriction \Delta(r)$ is a subset of ${}^{\Delta(r)}(U \setminus \{u\})$, and $e(r') = r$.*

**Theorem 3.3.** *Suppose that $\alpha > 2$ and $\mathcal{R}$ is the collection of db–relations (of dimension $\alpha$) with respect to some $u \in U$. Then,*

1. *If $r, s \in \mathcal{R}$, then $r \cap s \in \mathcal{R}$, and, if $\Delta(r) = \Delta(s)$, then $r \cup s \in \mathcal{R}$ and $r \setminus s \in \mathcal{R}$.*
2. *If $r \in \mathcal{R}$ and $i < \alpha$, then $c_i(r) \in \mathcal{R}$.*
3. *If $r \in \mathcal{R}$, then $s^i_j(r) \in \mathcal{R}$ for all $i, j < \alpha$.*

*Proof.* 1. and 2. are straightforward to prove, so we only show 3: Let $k < \alpha$, and suppose that $s^i_j(r) \cap v^u_k \neq \emptyset$; we need to show that $c_k(s^i_j(r)) = s^i_j(r)$. If $k = i$, then we are done, since $c_i(s^i_j(r)) = s^i_j(r)$. If $c_k(r) \neq r$, then $r \cap v^u_k = \emptyset$, i.e. $r \subseteq -v^u_k$, since $r$ is a db–relation. But then, $s^i_j(r) = c_i(r \cap d_{ij}) \leq c_i(r) \leq c_i(-v^u_k) = -v^u_k$, which contradicts our hypothesis; thus, $c_k(r) = r$. If $k \neq j$, then, by 1.5.8. of [13], we have $c_k(s^i_j(r)) = s^i_j(c_k(r)) = s^i_j(r)$. If $k = j$, then $c_j(r) = r$; furthermore, $r \cap d_{i,j} \cap v^u_k \neq \emptyset$, since $c_i(-v^u_j) = -v^u_j$. Hence, there is some $f \in r$ such that $f(j) = u = f(i)$, and therefore, $c_i(r) = r$. Again from 1.5.8. of [13] it follows that $s^i_j(r) = s^i_j(c_(r)) = c_i(r) = r$, and $c_j(s^i_j(r)) = c_j(r) = r$. □

The db - relations have some more simple but remarkable properties:

**Theorem 3.4.** *Let $r, s$ be non zero db - relations. Then,*

1. *If $r \subseteq s$, then $\Delta(s) \subseteq \Delta(r)$.*
2. *If $X, Y \subseteq \Delta(r)$ and $c_X(r) \subseteq c_Y(r)$, then $X \subseteq Y$.*
3. *If $X \subseteq \Delta(r)$, then $\Delta c_X(r) = \Delta r \backslash X$.*

A special case of Theorem 3.4.2 is that $c_i(r) = c_j(r)$ implies that $i = j$, whenever $r > 0$ and $i, j \in \Delta(r)$. This does not usually happen in arbitrary **Cs**; however, it seems necessary to model data base relations more realistically: Suppose that $r$ corresponds to a concrete relation of a database with scheme $X$, and that$i, j \in$ X. Now, $c_i$ corresponds to the projection of $r$ onto the scheme $X \backslash \{i\}$, similarly for $j$. If $i$ and $j$ name different columns of $r$, one would certainly want different projections of $r$ when one blanks out the columns $i$ and $j$ respectively. In other words, removing one column should not remove another column as well.

# 4 Data dependencies

A central concern of database design is to avoid redundancy and ensure the integrity of the database, and various *dependencies* have been considered. These may be viewed as specifications of the semantics of a database, and describe constraints among and within the data tables. For example, the earliest dependencies were the *functional dependencies* [5]: If $A_0, \ldots, A_k, B$ are attribute names and $r$ is a relation of the database, the functional dependency $A_0, \ldots, A_k \rightarrow_r B$ means that whenever two tuples from a relation $r$ agree on the attributes named by $A_0, \ldots, A_k$, then they agree also on attribute $B$. Since many such constraints are meaningful in various situations, many types of dependencies have been investigated; Thalheim [22] mentions in 1996 that over 100 have been considered in the literature.

Most of the research on dependency theory has centred around two main questions. The first of these is the *implication problem*for a given class of dependencies: For a given finite set $\Sigma$ of dependencies we say that $\Sigma$ implies a dependency $d$, and write $\Sigma \models d$, if every relation which satisfies all constraints in $\Sigma$ also satisfies $d$. The implication problem (a semantical concept) is to find an algorithm which decides $\Sigma \models d$. If there is such an algorithm, we say that the *implication problem is solvable* for the class of dependencies considered.

The second, closely related, problem is to find a complete axiom system for a class of dependencies, or else show that such a system does not exist. There is a large amount of literature on the subject - for an overview see e.g. [22] - which to explore in detail space does not permit.

In formulating constraints, the project and join operator play a significant role; indeed, the very general algebraic dependencies are stated in terms of these operators (see Table 1, where we assume that all expressions are well defined).

Table 1: Axioms for algebraic dependencies

| | |
|---|---|
| 1. $\pi_X(\pi_X(r)) = \pi_X(r),\ \pi_{a(r)}(r) = r.$ | 2. $r \bowtie \pi_X(r) = r,\ \pi_{a(r)}(r \bowtie s) \subseteq r$ |
| 3. $r \subseteq s \Rightarrow \pi_X(r) \subseteq \pi_X(s)$ | 4. $r \subseteq s \Rightarrow r \bowtie t \subseteq s \bowtie t$ |
| 5. $r \bowtie s = s \bowtie r$ | 6. $(r \bowtie s) \bowtie t = r \bowtie (s \bowtie t)$ |
| 7. $X \subseteq a(r)$ and $Y \subseteq a(s) \Rightarrow \pi_{X \cap Y}(r \bowtie s) \subseteq \pi_{X \cap Y}(r \bowtie \pi_Y(s))$ | |

By way of example, we shall concentrate in the sequel on general project–join dependencies and cylindric dependencies. We suppose that all data tables of a database have the same schema; this is no significant restriction by Theorem 3.1.

### 4.1 Project–join dependencies

Formally, a language for project–join expressions over a set of generators contains symbols for both operators and a set of predicates $\{P_j : j \in J\}$ all having the same finite arity, say, $\alpha$. The set of *project-join expressions over the generators* $\{P_j : j \in J\}$ (pjes) is defined as follows:

1. Each $P_j$ is a pje.
2. If $\tau$ is a pje and $Y \subseteq \alpha$, then $\pi_Y(\tau)$ is a pje.
3. If $\tau$ and $\sigma$ are pjes, then $\tau \bowtie \sigma$ is a pje.
4. No other expression is a pje.

Evaluation of pjes in databases is done in the obvious way: Suppose that $\mathcal{D} = \langle D_n, r_j \rangle_{n<\alpha, j\in J}$ is a database . If $\tau$ is a pje we define its value $e'(\tau)$ in $\mathcal{D}$ inductively:

- $e'(P_j) = r_j$
- $e'(\pi_Y \sigma) = \pi_Y(e'(\sigma))$
- $e'(\sigma \bowtie \rho) = e'(\sigma) \bowtie e'(\rho)$.

We can relate equations among pjes to equations in a reduct of $\mathsf{Cs}_\alpha$: A *(diagonal free) cylindric (lower) semilattice of dimension* $\alpha$, in short, a $\mathsf{csl}_\alpha$, is an algebra $\langle A, \cdot, c_i, \rangle_{i<\alpha}$ such that $\langle A, \cdot \rangle$ is a semilattice, and for all $x, y \in A$ and $i, j < \alpha$,

$$x \cdot c_i x = x \tag{C1}$$

$$c_i c_j x = c_j c_i x \tag{C2}$$

$$c_i(x \cdot c_i y) = c_i x \cdot c_i y. \tag{C3}$$

**Theorem 4.1.** *[9] The equational theory of pjes over $\alpha$–dimensional databases is equivalent to the equational theory of* $\mathsf{scsl}_\alpha$.

*Proof.* Consider the following translation tr of pjes onto cylindric expressions :

1. $\mathrm{tr}(P_j) = P_j$;
2. $\mathrm{tr}(\pi_Y \sigma) = c_{(\alpha \setminus Y)}\, \mathrm{tr}(\sigma)$;
3. $\mathrm{tr}(\sigma \bowtie \rho) = \mathrm{tr}(\sigma) \cdot \mathrm{tr}(\rho)$.

It is shown in [9] that for all pjes $\tau$ and $\sigma$, $\tau = \sigma$ holds in every $\alpha$–dimensional database if and only if $\mathrm{tr}(\tau) = \mathrm{tr}(\sigma)$ is valid in every $\mathsf{scsl}_\alpha$. □

A *project-join dependency* (pjd) is a formula $\tau = \sigma$ with $\tau$ and $\sigma$ pjes. A database $\mathcal{D}$ *satisfies* the pjd $\tau = \sigma$, written as $\mathcal{D} \models \tau = \sigma$, if and only if $e'(\tau) = e'(\sigma)$. A set $\Sigma$ of pjds *implies* a pjd $\delta$, written as $\Sigma \models \delta$ if and only if for every database $\mathcal{D}$, $\mathcal{D} \models \Sigma$ implies $\mathcal{D} \models \delta$. Note that we can express that a pje $\sigma$ follows from a pje $\tau$ ($\tau \leq \sigma$), since $e'(\tau) \subseteq e(\sigma)$ if and only if $e'(\tau \bowtie \sigma) = e'(\sigma)$.

Suppose that $\Sigma \cup \{\tau\}$ is a finite set of pjds. Since valid pjds are equivalent to valid equations of $\mathsf{scsl}_n$ by Theorem 4.1, it follows that the implication $\Sigma \models \tau$ is equivalent to a quasi–equation of cylindric expressions. The following result implies that there is no finite set of formulae axiomatizing $\alpha$–dimensional pjds if $\alpha > 2$:

**Theorem 4.2.** *[15]*
*if $\alpha > 2$, then the quasi-equational theory of* $\mathsf{scsl}_\alpha$ *is not finitely axiomatizable.*

## 4.2 Cylindric dependencies

Cylindric dependencies were introduced in [7], and they are an extension of project–join dependencies to the positive reduct of first order logic with equality. The corresponding query language $\mathcal{L}^+_\omega$ has the connectives $\wedge, \vee, \exists$, $\mathbf{T}, \mathbf{F}, =$, an infinite set $V = \{v_i : i < \omega\}$ of variables, and a set of predicates $\mathcal{P} = \{P_i : i \in I\}$; to simplify matters we may suppose that all $P_i$ have the same finite arity, say, $k$, using dummy embeddings. Variables are not typed, i.e. every variable is allowed to appear in any column of a predicate $P_i$.

In this setting, a database is a pair $\langle D, \mathcal{R}\rangle$, where $D$ is a nonempty set, and $\mathcal{R} = \{R_i : i \in I\}$ is a set of relations over $D$ with finite arity $\gamma(i)$.

Furthermore, the objects of the theory are valuations, i.e. mappings from the set $V$ of variables of the language to the domain $D$. A relation $r$ of arity $k$ of a database then corresponds to the set $\rho_r = \{f \in {}^V D : f[v_0, \ldots, v_{k-1}] \in r\}$.

Cylindrifications $c_i$ and diagonal elements $d_{ij}$ are defined as in part 3. above. A *cylindric dependency* now is just a positive equation of $Cs_\omega$, i.e. it does not contain complementation. The translation mapping $\mu$ from cylindric expressions, i.e. terms of positive $Cs'_\omega s$, to databases is essentially the same as the inverse that of [IL]. During this process, the constants $D_{ij}$, and ${}^V D$ are assumed implicitly to be elements of the database. The following axiom system is given:

C1. A set of equations which say that $\langle CE, \cdot, +, 0, 1\rangle$ is a distributive lattice with smallest element 0 and largest element 1.

C2. $c_i 0 = 0$

C3. $\tau \cdot c_i \tau = \tau$

C4. $c_i(\tau \cdot c_i \sigma) = c_i \tau \cdot c_i \sigma$

C5. $c_i(\tau + \sigma) = c_i \tau + c_i \sigma$

C6. $c_i c_j \tau = c_j c_i \tau$

C7. $d_{ii} = 1$

C8. If $j \neq i, k$, then $d_{ik} = c_j(d_{ij} \cdot d_{jk})$

C9. $c_j P_m = P_m$ for all $j \geq \gamma(m)$.

C10. $d_{ij} \cdot c_i(d_{ij} \cdot \tau) \leq \tau$.

for all $i, j, k \in \omega$, $m \in I$, and $\tau, \sigma \in CE$[1]. Implication of a CD $\varphi$ from a set of Cds is defined in the usual way. An abstract algebra $\mathfrak{C}$ which satisfies these axioms is called a *cylindric lattice with generators* $P_k$. If each element of $\mathfrak{C}$ is finite dimensional, we call $\mathfrak{C}$ *locally finite*. If the set of generators is understood or unimportant, we just speak of $\mathfrak{C}$ as a cylindric lattice. A

[1] Axiom C10 was actually omitted from [7],

cylindric lattice $\mathfrak{C}$ is called *representable* if it can be embedded into the product of algebras of the form

$$\langle 2^{^{\omega}D}, \cap, \cup, \emptyset, {}^{\omega}D, C_i, D_{ij} \rangle_{i,j \in \omega}.$$

Contrary to a claim in [7] the system C1 – C10 is not complete for the indended models. Indeed, the situation is more complicated:

**Theorem 4.3.** *[9] Let $\Sigma$ be a set of universal formulae axiomatizing (the quasi-equational theory of) representable cylindric lattices of dimension $\omega$. Then $\Sigma$ contains infinitely many variables.*

This result has the following consequence for cylindric dependencies:

**Corollary 4.4.** *There is no finite set of (universal) axiom schemas such that $\Sigma \models \tau$ iff $\Sigma \vdash \tau$ for every set $\Sigma \cup \{\tau\}$ of cylindric dependencies.*

There is no way to equationally repair the claim, since it is shown in [**?**] that, unlike representable cylindric algebras, the representable cylindric lattices do not form an equational class. Furthermore, it does not help to restrict ourselves to finite dimensions because of the following result:

**Theorem 4.5.** *[15] The quasi-equational and the equational theories of representable cylindric lattices of dimension $n$ is not finitely axiomatizable if $n > 2$.*

Note that in this case we have a stronger negative result than in the case of pjds, since we cannot finitely axiomatize even the valid equations between cylindric expressions, let alone valid inferences between equations.

# 5 Summary

In this chapter I have reviewed the connection between relational databases and cylindric structures. It turned out that there is a natural translation of relational databases into cylindric set lattices, and that Codd's relational operators have a simple interpretation in these structures. Consequently, various database constraints can be expressed in the equational language, and recent results in equational logic have been able to shed some light on the expressiveness and axiomatizability of these constraints.

## References


[1] *28th Annual Symposium on Foundations of Computer Science, 12-14 October 1987, Los Angeles, California, USA*. IEEE, 1987.

[2] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal of Computing*, 8:218–246, 1979.

[3] S.A. Bukhari. A relation algebra model for data bases. Research report CS-75-11, Department of Computer Science, University of Waterloo, 1975.

[4] E. F. Codd. A relational model of data for large shared databanks. *Comm. of the ACM*, 13:377–387, 1970.

[5] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[6] Stavros Cosmodakis. Equational theories and database constraints. Doctoral dissertation, Massachusetts Institute of Technology, 1985.

[7] Stavros Cosmodakis. Database theory and cylindric lattices. In *IEEE 28th Symposion on the Foundations of Computer Science*, pages 411–420, 1987.

[8] Ivo Düntsch. A note on cylindric lattices. In *Algebraic Methods in Logic and Computer Science*, pages 231–238. Banach Center Publications 28, Warsaw, 1993.

[9] Ivo Düntsch and Szabolcs Mikulás. Cylindric structures and dependencies in relational databases. *Theoretical Computer Science*, 269:451–468, 2001.

[10] Ron Fagin. Horn clauses and data dependencies. *Journal of the ACM*, 29:952–985, 1982.

[11] Ronald Fagin and Moshe Y. Vardi. The theory of data dependencies - a survey. In M. Anshel and W. Gewirtz, editors, *Mathematics of Information Processing*, pages 19–71. AMS, 1986.

[12] Hans W. Guesgen and Peter B. Ladkin. An algebraic approach to general boolean constraint problems. Technical Report TR-90-008, International Computer Science Institute, Berkeley, CA, March 1990.

[13] Leon Henkin, J. Donald Monk, and Alfred Tarski. *Cylindric algebras, Part I.* North Holland, Amsterdam, 1971.

[14] Leon Henkin, J. Donald Monk, and Alfred Tarski. *Cylindric algebras, Part II.* North Holland, Amsterdam, 1985.

[15] I. Hodkinson and S Mikulás. Axiomatizability of reducts of algebras of relations. *Algebra Universalis*, 43:127–156, 2000.

[16] Tomasz Imielinski and Witold Lipski. On the undecidability of equivalence problems for relational expressions. In H. Gallaire and J. M. Nicolas, editors, *Advances in Database Theory Vol. 2*, pages 393–409. Plenum Press, 1984.

[17] Tomasz Imilienski and Witold Lipski. The relational model of data and cylindric algebras. *Journal of Computer and System Sciences*, 28:80–102, 1984.

[18] Ryszard Janicki and Ridha Khedri. On a formal semantics of tabular expressions. *Science of Computer Programming*, 2000. To appear.

[19] Paris C. Kanellakis. Elements of relational database theory. Technical Report CS-89-39, Department of Computer Science, Brown University, 1989.

[20] István Németi. Algebraizations of quantifier logics. *Studia Logica*, 50:485–569, 1991. Updated version 12.1 (January 1997) is available from `ftp.math-inst.hu/pub/algebraic-logic/survey.ps`.

[21] Silvija Seres. *The Algebra of Logic Programming.* PhD thesis, Wolfson College, Oxford University, 2001.

[22] Bernhard Thalheim. An overview on semantical constraints for database models. In *Proceedings of The 6th International Conference on Intellectual Systems and Computer Science*, pages 1–10, 1996.

[23] Jan Van den Bussche. Applications of alfred tarski's ideas in database theory. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France,*, volume 2142 of *Lecture Notes in Computer Science*, pages 20–37. Springer Verlag, 2001.

[24] M. Yannakakis and C. H. Papadimitriou. Algebraic dependencies. *Journal of Computer and System Sciences*, 25:80–102, 1982.