Functional dependency theory made ’simpler’

J.N. Oliveira

`jno@di.uminho.pt`

Techn. Report DI-PURe-05.01.01

2005, January

PURE
Program Understanding and Re-engineering: Calculi and Applications
(Project `POSI/ICHS/44304/2002`)

Departamento de Informática da Universidade do Minho
Campus de Gualtar — Braga — Portugal

DI-PURe-05.01.01
Functional dependency theory made ’simpler’ by J.N. Oliveira

Abstract

In the context of Joost Visser’s *Spreadsheets under Scrutiny* talk, I have been looking at refinement laws for spreadsheet normalization. Back to good-old ‘functional dependency theory’, D. Maier’s book etc, I ended up by rephrasing of the standard theory using the binary relation pointfree calculus. It turns out that the theory becomes simpler and more general, thanks to the calculus of ’simplicity’ and coreflexivity. This research also shows the effectiveness of the *binary* relation calculus in “explaining” and reasoning about the n-ary relation calculus “à la Codd”.

1 What is a functional dependency?

In the standard relational database technology, objects are recorded by assigning values to their observable properties or *attributes*. A database file is a collection of attribute assignments, one per object. Displayed in a bi-dimensional tabular format, each object corresponds to a *tuple* of values, or *row* — eg. row 10 in a Excel spreadsheet — and each column lists the values of a particular attribute in all tuples (eg. row ”A” in a Excel spreadsheet). All values of a particular attribute (say i) are of the same type (say A_i). For n such attributes, a relational database file R can be regarded as a set of n-tuples, that is, $R \subseteq A_1 \times \ldots \times A_n$.

Attribute names normally replace natural numbers in the identification of attributes. The enumeration of all attribute names in a database relation, for instance

$$S = \{\textsc{Pilot}, \textsc{Flight}, \textsc{Date}, \textsc{Departs}\} \tag{1}$$

concerning an airline scheduling system [1], is a set called the relation's *scheme*. This scheme captures the *syntax* of the data. What about *semantics*?

Even non-experts in airline scheduling will accept the following “business” rule: *A single pilot is assigned to a given flight, on a given date.*

This restriction is an example of a so-called *functional dependency* (FD) among attributes, which can be stated more formally as follows: *attribute* PILOT *is functionally dependent on* FLIGHT, DATE. In the standard practice, this will be abbreviated by

$$\textsc{Flight Date} \rightarrow \textsc{Pilot}$$

which people also read as follows: FLIGHT DATE *functionally determines* PILOT.

The addition of functional dependencies to a relational schema is comparable to the addition of axioms to an algebraic signature (eg. axioms such as $pop(push(a, s) = s$ adding semantics to the syntax of a stack datatype involving operators $push$ and pop). How do we reason about such functional dependencies? Can we simplify a set of dependencies by removing the redundant ones, if any? How do we design a concrete database implementation from a relational schema *and* its dependencies?

The information system community is indebted to Codd for his pioneering work on the very foundations of the *relational data model* theory [Cod70]. Since then, relational database theory has been thoroughly studied, and several textbooks are available on the topic, namely [Mai83], [Ull88] and [GUW02].

[1] This example is taken from [Mai83].

At the heart of relational database theory we find *functional dependency* (FD) *theory*, which is axiomatic in nature and stems from the definition of FD-satisfiability which follows.

Definition 1. *Given subsets $x, y \subseteq S$ of the relation scheme S of a relation R, this relation is said to satisfy functional dependency $x \to y$ iff all pairs of tuples $t, t' \in R$ which "agree" on x also "agree" on y, that is,*

$$\langle \forall\, t, t' \;:\; t, t' \in R :\;\; t[x] = t'[x] \Rightarrow t[y] = t'[y] \,\rangle \tag{2}$$

(Notation $t[x]$ meaning "the values in t of the attributes in x" will be scrutinized in the sequel.) □

The closure of a set of FDs is based on the so-called *Armstrong axioms* [Mai83] which can be used as inference rules for FDs. An "equivalent" set of axioms has been found which turns FD checking more efficient.

Why has this theory "gone this way"? I don't know; but, perhaps one reason lays in the fact that formula (2), with its logical implication inside a "two-dimensional" universal quantification, is not particularly agile. Designs involving several FDs at the same time can be hard to reason about.

This calls for a simplification of this very basis of FD-theory. The main purpose of this report is to present an alternative, more general setting for FD-theory based on the pointfree *binary* relation calculus. It turns out that the theory becomes more general and considerably simpler, thanks to the calculus of *simplicity* and *coreflexivity*. (Details about this terminology will be presented shortly.)

We will start by reviewing some basic principles. Note that — of course — the qualifier "functional" in "functional dependency" stems from "function". So our first efforts go into making sure we have a clear idea of "what a function is".

2 What is a function? —the Leibniz view

Functions are special cases of binary relations satisfying two main properties:

– *"Left" Uniqueness*

$$b\ f\ a \ \wedge\ b'\ f\ a \Rightarrow b = b' \tag{3}$$

– *Leibniz principle*

$$a = a' \Rightarrow f\ a = f\ a' \tag{4}$$

It can be shown (see an exercise later on) that this is the same as saying that functions are *simple* and *entire* relations, respectively:

– f is simple:

$$\mathit{img}\ f \subseteq id \tag{5}$$

– f is entire:

$$id \subseteq \mathit{ker}\ f \tag{6}$$

Formulæ (5) and (6) are examples of *pointfree* notation in which *points* (eg. a, a', b, b') disappear. (For instance, instead of writing $a = a'$, we identify the identity relation id which relates a and a' when they are the same.) In order to parse such *compressed* formulæ we need to understand the meaning of expressions *ker* f (read “the *kernel* of f”) and *img* f (read “the *image* of f),

$$\mathit{ker}\ R = R^\circ \cdot R \tag{7}$$

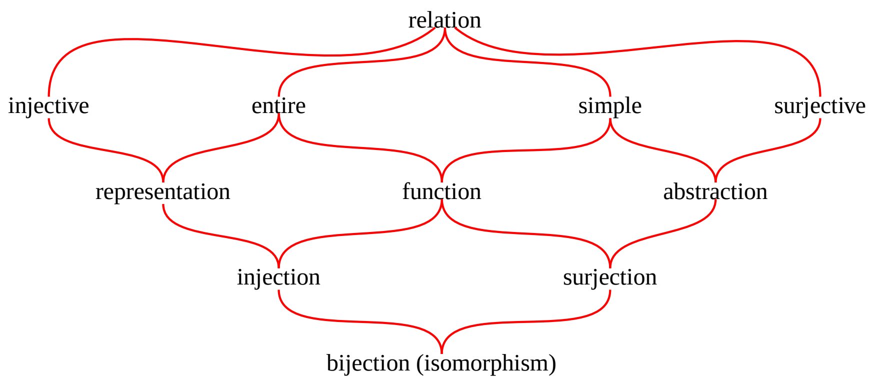
$$\mathit{img}\ R = R \cdot R^\circ \tag{8}$$

whose definitions involve two basic relational combinators: converse (R°) and composition ($R \cdot S$). The former converts a relation R into R° such that $a(R^\circ)b$ holds iff bRa holds. (We write bRa to mean that pair $\langle b, a\rangle$ is in R.) The latter *(composition)* is defined in the usual way: $b(R \cdot S)c$ holds wherever there exist one or more mediating $a \in A$ such that $bRa \wedge aSc$, where $B \xleftarrow{R} A$ and $C \xleftarrow{R} B$ are two binary relations on datatypes A (source) and B (target) and B (source) and C (target), respectively. As in (5) and (6), the underlying partial order on relations is written $R \subseteq S$, meaning

$$R \subseteq S \equiv \langle \forall\, b, a \ :: \ bRa \Rightarrow bSa \rangle \tag{9}$$

for all a and b suitably typed.

As a matter of fact, what we have just presented is part of a wider binary relation taxonomy,



whose four top-level classification criteria are captured by the following table,

	Reflexive	*Coreflexive*
ker R	entire R	injective R
img R	surjective R	simple R

where R is said to be *reflexive* iff it is at least the identity ($id \subseteq R$) and it is said to be *coreflexive* (or a *partial identity*) iff it is at most the identity, that is, if $R \subseteq id$ holds.

Coreflexive relations are fragments of the identity relation which can be used to model predicates or sets. The meaning of a *predicate* p is the coreflexive $[\![p]\!]$ such that $b[\![p]\!]a \equiv (b = a) \ \wedge \ (p\ a)$. This is the relation that maps every a which satisfies p (and only such a) onto itself. The meaning of a *set* $S \subseteq A$ is $[\![\lambda a.a \in S]\!]$, that is,

$$b[\![S]\!]a \equiv (b = a) \ \wedge \ a \in S \tag{10}$$

Wherever clear from the context, we will drop brackets $[\![\ ]\!]$.

Before we embark on converting (2) into pointfree notation, let us see an alternative view of functions better suited for calculations.

3 What is a function? —the "Galois view"

Shunting rules. To say that f *is a function* is *equivalent* to stating any of the Galois connections

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S \tag{11}$$
$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f \tag{12}$$

Let us check one of these, say (11). (More about this can be found in eg. [Hoo97,BdM97,Bac04] etc.)

That f entire and simple *implies* equivalence (11) can be proved by mutual implication:

$$
\begin{array}{ll}
 & f \cdot R \subseteq S \\
\Rightarrow & \quad \{ \text{ monotonicity of composition } \} \\
 & f^\circ \cdot f \cdot R \subseteq f^\circ \cdot S \\
\Rightarrow & \quad \{ \ f \text{ is entire (6) } \} \\
 & R \subseteq f^\circ \cdot S \\
\Rightarrow & \quad \{ \text{ monotonicity of composition } \}
\end{array}
$$

$$
\begin{array}{ll}
 & f \cdot R \subseteq f \cdot f^\circ \cdot S \\
\Rightarrow & \quad \{ \ f \text{ is simple (5)} \ \} \\
 & f \cdot R \subseteq S
\end{array}
$$

That (11) *implies* that f is entire and simple is easy to see: let $R, S := id, f \cdot R$ (left-cancellation) or $S, R := id, f^\circ \cdot S$ (right-cancellation), respectively.

Function converses enjoy a number of properties of which the following is singled out because of its rôle in pointwise-pointfree conversion [BB03] :

$$b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a) \tag{13}$$

The use of (13) to convert (3, 4) into (5, 6), respectively, is left as an exercise.

4 FD-satisfiability in pointfree style

Attributes are functions. Let R be a relation with schema S, t a tuple in R and a be an attribute in S. Notation $t[a]$ was adopted in (2) to mean "the value of attribute a in t". This indicates that a can be identified with the *projection* function which extracts the value which t exhibits as property a. Since this extends to a collection x of attributes, we can convert (2) into

$$\langle \forall\ t, t' \ : \ t, t' \in R : \ (x\ t) = (x\ t') \Rightarrow (y\ t) = (y\ t') \rangle$$

Assuming the universal quantification implicit, we reason:

$$
\begin{array}{ll}
 & t \in R \ \wedge \ t' \in R \ \wedge \ (x\ t) = (x\ t') \Rightarrow (y\ t) = (y\ t') \\
\equiv & \quad \{ \ \text{(13) twice, for } R := id \ \} \\
 & t \in R \ \wedge \ t' \in R \ \wedge \ t(x^\circ \cdot x)t' \Rightarrow t(y^\circ \cdot y)t' \\
\equiv & \quad \{ \ \text{(10) twice} \ \} \\
 & t[\![R]\!]u \ \wedge \ t = u \ \wedge \ t'[\![R]\!]u' \ \wedge \ t' = u' \ \wedge \ t(x^\circ \cdot x)t' \Rightarrow t(y^\circ \cdot y)t' \\
\equiv & \quad \{ \ \wedge \text{ is commutative; substitution of equals for equals; converse} \} \\
 & t[\![R]\!]u \ \wedge \ u(x^\circ \cdot x)u' \ \wedge \ u'[\![R]\!]^\circ t' \Rightarrow t(y^\circ \cdot y)t' \\
\equiv & \quad \{ \ \text{relation inclusion (9)} \ \} \\
 & [\![R]\!] \cdot (x^\circ \cdot x) \cdot [\![R]\!]^\circ \subseteq y^\circ \cdot y \qquad (14) \\
\equiv & \quad \{ \ \text{shunting rules (11) and (12)} \ \} \\
 & y \cdot [\![R]\!] \cdot x^\circ \cdot x \cdot [\![R]\!]^\circ \cdot y^\circ \subseteq id
\end{array}
$$

$\equiv$ { converse versus composition, $(R \cdot S)^\circ = S^\circ \cdot R^\circ$, followed by (8) }

$\quad img\,(y \cdot [\![R]\!] \cdot x^\circ) \subseteq id$

In summary: a relation R as in definition 1 satisfies functional dependency $x \to y$ iff the binary relation

$$y \cdot [\![R]\!] \cdot x^\circ \tag{15}$$

is simple, cf. (5).

5 Functional dependencies in general

Our definition of FD starts from the observation that relation R and projection functions x and y in (15) can be generalized to arbitrary binary relations and functions. This leads to the more general definition which follows. (The use of "$\rightharpoonup$" instead of "$\to$" is intentional — it stresses the move from the restricted to the generic notion.)

Definition 2. *We say that relation* $B \xleftarrow{R} A$ *satisfies the "*$f \rightharpoonup g$*" functional dependency — written* $f \overset{R}{\rightharpoonup} g$ *— iff* $g \cdot R \cdot f^\circ$ *in*

$$\begin{tikzcd} A \arrow[d, "g"'] & B \arrow[l, "R"'] \arrow[d, "f"] \\ C & D \arrow[l, "g \cdot R \cdot f^\circ"] \end{tikzcd}$$

is simple. Equivalent definitions are

$$f \overset{R}{\rightharpoonup} g \equiv R \cdot (\mathit{ker}\, f) \cdot R^\circ \subseteq \mathit{ker}\, g \tag{16}$$

— cf. (14) and (7) — and

$$f \overset{R}{\rightharpoonup} g \equiv \mathit{ker}\,(f \cdot R^\circ) \subseteq \mathit{ker}\, g \tag{17}$$

since converse commutes with composition,

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{18}$$

Function f *(resp.* g*) will be mentioned as the* left side *or* antecedent *(resp.* right side *or* consequent*) of FD* $f \overset{R}{\rightharpoonup} g$. $\Box$

Examples. The reader may wish to check that $f \overset{R}{\rightharpoonup} g$ holds for R any of the relations tabulated by the a and b columns of

a	b	$f\ b = b^2$	$g\ a = rem\ a\ 3$
2	-1	1	2
5	-1	1	2
17	1	1	2
10	-2	4	1
4	-2	4	1
1	2	4	1

and

a	b	$f = id$	$g = \pi_1$
$(1,2)$	-1	-1	1
$(1,10)$	-1	-1	1
$(0,0)$	1	1	0
$(5,6)$	-2	-2	5
$(5,0)$	-2	-2	5
$(1,2)$	2	2	1

Basic properties. In contrast with (2), equations (16) and (17) are easy to reason about, as the reader may check by proving the following, elementary properties, which hold for all R, f, g of appropriate type:

$$f \overset{\perp}{\rightharpoonup} g \tag{19}$$

($\perp$ denotes the empty relation)

$$f \overset{R}{\rightharpoonup} ! \tag{20}$$

(where $1 \xleftarrow{!} A$ denotes the unique, "everywhere 'nothing' " function of its type)

$$id \overset{R}{\rightharpoonup} id \equiv \ R \text{ is simple} \tag{21}$$

$$f \overset{R}{\rightharpoonup} f \Leftarrow R \subseteq id \tag{22}$$

An immediate consequence of (22) is

$$f \overset{id}{\rightharpoonup} f \tag{23}$$

6 The role of injectivity

Ordering relations by injectivity. It can be observed that what matters about f and g in (16) is their "degree of injectivity" as measured by *ker* f and *ker* g, in opposite directions: more injective f and less injective g will strengthen a given FD $f \overset{R}{\rightharpoonup} g$. An extreme case is $f = id$ and $h = !$ — functional dependency $id \overset{R}{\rightharpoonup} !$ will always hold, for any R, as cf. (20).

In order to *measure* injectivity in general we define the *injectivity preorder* on relations as follows:

$$R \leq S \equiv \mathit{ker}\ S \subseteq \mathit{ker}\ R \tag{24}$$

that is, $R \leq S$ means R is *less* injective than S. Note that R and S must have the same source but don't need to share the same target datatype. For instance, it is easy to see that

$$! \leq R \tag{25}$$

$$R \leq \bot \tag{26}$$

hold, since the kernel of ! is the top relation and that of the empty relation is empty.

Which relational operators respect the injectivity preorder? It is easy to see that pre-composition does,

$$R \leq S \Rightarrow R \cdot T \leq S \cdot T \tag{27}$$

as can be easily proved:

$$\begin{array}{ll}
 & R \leq S \\
\equiv & \quad \{ \text{ (24) and (7) } \} \\
 & S^\circ \cdot S \subseteq R^\circ \cdot R \\
\Rightarrow & \quad \{ \text{ monotonicity of } (T^\circ \cdot \ ) \text{ and } (\ \cdot T) \} \\
 & T^\circ \cdot S^\circ \cdot S \cdot T \subseteq T^\circ \cdot R^\circ \cdot R \cdot T \\
\equiv & \quad \{ \ (S \cdot R)^\circ = R^\circ \cdot S^\circ \text{ (twice), followed by (7 and (24)) } \} \\
 & R \cdot T \leq S \cdot T
\end{array}$$

Relators exhibit the same monotonic behaviour:

$$R \leq S \Rightarrow \mathsf{F}\, R \leq \mathsf{F}\, S \tag{28}$$

cf.

$$\begin{array}{ll}
 & \mathsf{F}\, R \leq \mathsf{F}\, S \\
\equiv & \quad \{ \text{ definition (24) } \} \\
 & \mathit{ker}\,(\mathsf{F}\, S) \subseteq \mathit{ker}\,(\mathsf{F}\, R) \\
\equiv & \quad \{ \ \mathit{ker}\,(\mathsf{F}\, R) = \mathsf{F}(\mathit{ker}\, R) \ \} \\
 & \mathsf{F}(\mathit{ker}\, S) \subseteq \mathsf{F}(\mathit{ker}\, R) \\
\Leftarrow & \quad \{ \text{ relators are monotonic } \} \\
 & \mathit{ker}\, S \subseteq \mathit{ker}\, R \\
\equiv & \quad \{ \text{ (24) again } \} \\
 & R \leq S
\end{array}$$

FD defined via the injectivity ordering. The close relationship between FDs and injectivity of observations is well captured by the following re-statement of (17) in terms of (24):

$$f \overset{R}{\rightharpoonup} g \;\equiv\; g \leq f \cdot R^{\circ} \tag{29}$$

For its conciseness, this definition of FD is very amenable to calculation. Such is the case of the proof that two FDs with matching antecedent / consequent functions yield a composite FD,

$$f \overset{S \cdot R}{\rightharpoonup} h \Leftarrow f \overset{R}{\rightharpoonup} g \;\wedge\; g \overset{S}{\rightharpoonup} h \tag{30}$$

which follows:

$$
\begin{array}{cl}
 & f \overset{R}{\rightharpoonup} g \;\wedge\; g \overset{S}{\rightharpoonup} h \\
\equiv & \qquad \{ \text{ (29) twice } \} \\
 & g \leq f \cdot R^{\circ} \;\wedge\; h \leq g \cdot S^{\circ} \\
\Rightarrow & \qquad \{ \leq\text{-monotonicity of } (\;\cdot S^{\circ}) \text{ (27) followed by } (S \cdot R)^{\circ} = R^{\circ} \cdot S^{\circ} \} \\
 & g \cdot S^{\circ} \leq f \cdot (S \cdot R)^{\circ} \;\wedge\; h \leq g \cdot S^{\circ} \\
\Rightarrow & \qquad \{ \leq\text{-transitivity } \} \\
 & h \leq f \cdot (S \cdot R)^{\circ} \\
\equiv & \qquad \{ \text{ (29) again } \} \\
 & f \overset{S \cdot R}{\rightharpoonup} h
\end{array}
$$

A category of functions. Note in passing that (30) and (23) suggest that we can build a category whose objects are functions f, g, etc. and whose arrows $f \xrightarrow{R} g$ are relations which satisfy $f \overset{R}{\rightharpoonup} g$.

Simultaneous observations. In the same way x and y in (2) may involve more that one observable attribute, we would like f and g in (16) to involve more than one *observation* function. More observations add more detail and so are likely to be more injective. The relational *split* combinator

$$(a, b)\langle R, S\rangle c \equiv a\, R\, c \;\wedge\; b\, S\, c \tag{31}$$

captures this effect, and facts

$$R \leq \langle R, S\rangle \text{ and } S \leq \langle R, S\rangle$$

are easy to check by recalling

$$\mathit{ker}\,\langle R, S\rangle = (\mathit{ker}\,R) \cap (\mathit{ker}\,S) \tag{32}$$

which stems from

$$\langle R, S\rangle^{\circ} \cdot \langle X, Y\rangle = (R^{\circ} \cdot X) \cap (S^{\circ} \cdot Y) \tag{33}$$

Moreover, the following Galois connection

$$\langle R, S\rangle \leq T \equiv R \leq T \ \wedge\ S \leq T \tag{34}$$

stems from the one underlying $\cap$:

$$\begin{array}{ll}
& \langle R, S\rangle \leq T \\
\equiv & \qquad \{ \text{ (24) and (32) } \} \\
& \mathit{ker}\,T \subseteq (\mathit{ker}\,R) \cap (\mathit{ker}\,S) \\
\equiv & \qquad \{ \ \cap\text{-universal property (GC) } \} \\
& \mathit{ker}\,T \subseteq \mathit{ker}\,R \ \wedge\ \mathit{ker}\,T \subseteq \mathit{ker}\,S \\
\equiv & \qquad \{ \text{ (24) twice } \} \\
& R \leq T \ \wedge\ S \leq T
\end{array}$$

The anti-symmetric closure of $\leq$ yields an equivalence relation

$$R \simeq S \equiv \mathit{ker}\,R = \mathit{ker}\,S \tag{35}$$

which is such that, for instance, $! \simeq \top$ holds. The following equivalences will be relevant in the sequel, for suitably typed R, S and T:

$$R \simeq \langle R, R\rangle \tag{36}$$
$$\langle R, S\rangle \simeq \langle S, R\rangle \tag{37}$$
$$\langle T, \langle R, S\rangle\rangle \simeq \langle\langle T, R\rangle, S\rangle \tag{38}$$

Function injectivity. Since attributes in the relational database model are functions, we will be particularly interested in comparing functions for their injectivity. Note that the kernel of a function is an equivalence relation and thus always reflexive. So, restricted to functions, the $\leq$ ordering is such that, for all f,

$$! \leq f \leq id \tag{39}$$

and

$$f \simeq id \equiv f \text{ is an injection} \tag{40}$$

Also note the equivalence

$$f \leq g \equiv g \xrightarrow{id} f \tag{41}$$

From (39) and (27) we obtain $f \cdot R \leq R$. From (5) we draw $id \leq f^\circ$ and then $R \leq f^\circ \cdot R$. Putting these together:

$$f \cdot R \leq R \leq f^\circ \cdot R \tag{42}$$

FDs on functions. As special cases of relations, functions may also satisfy functional dependencies. For instance, it will be easy to show that $bagify \xrightarrow{setify} id$ holds, where $bagify$ (resp. $setify$) is the function which extracts, from a finite list, the bag (resp. set) of all its elements.

More generally, the following Galois connection

$$R \cdot h \leq S \equiv R \leq S \cdot h^\circ \tag{43}$$

holds, which can be regarded as an "injectivity variant" of "shunting" rules (11,12). From (43) we draw:

$$g \cdot h \leq f \equiv f \xrightarrow{h} g \equiv g \leq f \cdot h^\circ \tag{44}$$

On $\simeq$-equivalence. The discrimination of functions beyond $\simeq$-equivalence is unnecessary in the context of FD reasoning. The reader will observe this by checking (30) once the second occurrence of g is replaced by some j such that $g \simeq j$: the proof is essentially the same, since $ker\, g = ker\, j$. Thus the more general pattern of FD chaining which follows:

$$f \xrightarrow{S \cdot R} h \Leftarrow f \xrightarrow{R} g \ \wedge \ g \simeq j \ \wedge \ j \xrightarrow{S} h \tag{45}$$

Since ordering and repetition in "splits" are $\simeq$-irrelevant — recall (36), (37) and (38) — we will abbreviate $\langle f, g \rangle$ by fg, or by gf, wherever this notation shorthand is welcome and makes sense [2]. Such is the case of a fact which will prove particularly useful in the sequel:

$$f \xrightarrow{R} gh \equiv f \xrightarrow{R} g \ \wedge \ f \xrightarrow{R} h \tag{46}$$

[2] This is inspired by a similar shorthand popular in the standard notation of relational database theory: attribute set union is denoted by simple juxtaposition.

The proof of (46) is as follows:

$$
\begin{array}{ll}
& f \overset{R}{\rightharpoonup} gh \\
\equiv & \qquad \{ \text{ (29) ; expansion of shorthand } gh \ \} \\
& \langle g, h \rangle \leq f \cdot R^{\circ} \\
\equiv & \qquad \{ \text{ (34) } \} \\
& g \leq f \cdot R^{\circ} \ \wedge \ h \leq f \cdot R^{\circ} \\
\equiv & \qquad \{ \text{ (29) twice } \} \\
& f \overset{R}{\rightharpoonup} g \ \wedge \ f \overset{R}{\rightharpoonup} h
\end{array}
$$

FD strengthening. The comment above about the contra-variant behaviour (concerning injectivity) of the antecedent and consequent functions of an FD is now made precise,

$$
h \overset{R}{\rightharpoonup} k \Leftarrow h \geq f \ \wedge \ f \overset{R}{\rightharpoonup} g \ \wedge \ g \geq k \tag{47}
$$

whose proof s immediate:

$$
\begin{array}{ll}
& h \geq f \ \wedge \ f \overset{R}{\rightharpoonup} g \ \wedge \ g \geq k \\
\equiv & \qquad \{ \text{ (41) twice } \} \\
& h \overset{id}{\rightharpoonup} f \ \wedge \ f \overset{R}{\rightharpoonup} g \ \wedge \ g \overset{id}{\rightharpoonup} k \\
\Rightarrow & \qquad \{ \text{ (30) twice; identity of composition } \} \\
& h \overset{R}{\rightharpoonup} k
\end{array}
$$

The following are corollaries of (47), since $fh \geq f$:

$$
fh \overset{R}{\rightharpoonup} g \Leftarrow f \overset{R}{\rightharpoonup} g \tag{48}
$$

$$
f \overset{R}{\rightharpoonup} g \Leftarrow f \overset{R}{\rightharpoonup} gh \tag{49}
$$

By $\Leftarrow$-transitivity, we see that it is always possible in a FD to move observations from the consequent ("dependent") side to the antecedent ("independent") one:

$$
fh \overset{R}{\rightharpoonup} g \Leftarrow f \overset{R}{\rightharpoonup} gh \tag{50}
$$

Moving the "very last" one also makes sense, since

$$
fhg \overset{R}{\rightharpoonup} ! \Leftarrow fh \overset{R}{\rightharpoonup} g
$$

7 Keys

Every x such that $x \overset{R}{\rightharpoonup} id$ — if it exists — is called a *superkey* for R. Keys are minimal superkeys, that is, they are functions x as above such that, for all $y \leq x$, $y \overset{R}{\not\rightharpoonup} id$. In other words,

$$x \text{ is a key of } R \equiv x \overset{R}{\rightharpoonup} id \;\wedge\; \langle \forall\, y \;:\; y \overset{R}{\rightharpoonup} id :\; y \not\leq x \rangle$$

From (22) and (39) we draw that id is always a (maximal) superkey.

8 The Armstrong-axioms

In this section we prove the correctness of the Armstrong-axioms [Mai83], which are the standard inference rules for FDs underlying relational database theory. We show that FD theory is a natural consequence of the pointfree formalization presented earlier on.

In the standard formulation, these axioms involve subsets of attributes of a relational schema S ordered by inclusion, eg. $X \subseteq Y \subseteq S$. Unions of such attribute subsets are written by juxtaposition, eg. XY instead of $X \cup Y$. Since attributes $X$and Y "are" (projection) functions, XY will mean the *split* of such projections. In our setting, we generalize these to arbitrary functions ordered by injectivity. In fact, it is easy to see that $X \subseteq Y$ implies $X \leq Y$. (For notation economy, we use the same symbols X and Y to denote *both* the attribute symbol and the associated projection.) The whole schema S corresponds to a maximal observation. In our setting, this is captured by the identity id, since — by product reflexion — the *split* of all projections in a finite product is the identity. (This observation will be made more precise in section 8.2.)

As we have seen, n-ary relational database tables are sets of tuples which we model by coreflexive relations. For instance, a table with three attributes $T \subseteq A \times B \times C$ will be modelled by coreflexive

$$A \times B \times C \xleftarrow{[\![T]\!]} A \times B \times C$$

such that $t[\![T]\!]t' \equiv t = t' \;\wedge\; t \in T$. In this section, we will abbreviate $[\![T]\!]$ by T.

Proofs of the Armstrong-axioms follow:

– (A1) **Reflexivity** :

$$y \leq x \Rightarrow x \overset{T}{\rightharpoonup} y \tag{51}$$

This follows from $x \overset{T}{\rightharpoonup} x \ \wedge \ x \geq y$, recall (22) and (47). Another way to put it is (let $x := yz$)

$$yz \overset{T}{\rightharpoonup} y \tag{52}$$

– (A2) **Augmentation** :

$$x \overset{T}{\rightharpoonup} y \Rightarrow xz \overset{T}{\rightharpoonup} yz \tag{53}$$

Proof:

$$\begin{array}{ll}
 & xz \overset{T}{\rightharpoonup} yz \\
\equiv & \qquad \{ \text{ (46) } \} \\
 & xz \overset{T}{\rightharpoonup} y \ \wedge \ xz \overset{T}{\rightharpoonup} z \\
\equiv & \qquad \{ \text{ Reflexivity (A1) in version (52) } \} \\
 & xz \overset{T}{\rightharpoonup} y \\
\Leftarrow & \qquad \{ \text{ (48) } \} \\
 & x \overset{T}{\rightharpoonup} y
\end{array}$$

(The implication just above is Maier's version of this axiom [Mai83].)

– (A3) **Transitivity** :

$$x \overset{T}{\rightharpoonup} y \ \wedge \ y \overset{T}{\rightharpoonup} z \Rightarrow x \overset{T}{\rightharpoonup} z \tag{54}$$

This stems from (30) for S and R the same coreflexive T, in which case $T \cdot T = T$.

8.1 The Armstrong-axioms — alternative version

– (A4) **Additivity** (or **Union**):

$$x \overset{T}{\rightharpoonup} y \ \wedge \ x \overset{T}{\rightharpoonup} z \Rightarrow x \overset{T}{\rightharpoonup} yz \tag{55}$$

This is one of the "ping-pong" sides of (46).

– (A4') **Projectivity**:

$$x \overset{T}{\rightharpoonup} yz \Rightarrow x \overset{T}{\rightharpoonup} y \ \wedge \ x \overset{T}{\rightharpoonup} z \tag{56}$$

This is the other side of (46).

– (A5) **Pseudo-transitivity** :

$$x \overset{T}{\rightharpoonup} y \;\wedge\; wy \overset{T}{\rightharpoonup} z \Rightarrow xw \overset{T}{\rightharpoonup} z \tag{57}$$

This stems from the first version of the axioms alone:

$$\begin{aligned} & x \overset{T}{\rightharpoonup} y \;\wedge\; wy \overset{T}{\rightharpoonup} z \\ \Rightarrow \quad & \{ \text{ augmentation (A2)} \} \\ & xw \overset{T}{\rightharpoonup} yw \;\wedge\; wy \overset{T}{\rightharpoonup} z \\ \Rightarrow \quad & \{ \text{ transitivity (A3)} \} \\ & xw \overset{T}{\rightharpoonup} z \end{aligned}$$

– (A6) **Decomposition** :

$$x \overset{T}{\rightharpoonup} y \;\wedge\; z \leq y \Rightarrow x \overset{T}{\rightharpoonup} z \tag{58}$$

This is (47) for $f = k$. Alternatively,

$$\begin{aligned} & x \overset{T}{\rightharpoonup} y \;\wedge\; z \leq y \\ \Rightarrow \quad & \{ \text{ (A1) } \} \\ & x \overset{T}{\rightharpoonup} y \;\wedge\; y \overset{T}{\rightharpoonup} z \\ \Rightarrow \quad & \{ \text{ (A3) } \} \\ & x \overset{T}{\rightharpoonup} z \end{aligned}$$

– (A7) **Reflexivity** :

$$x \overset{T}{\rightharpoonup} x \tag{59}$$

This is (22) or (A1) for $y = x$.

– (A8) **Accumulation** :

$$x \overset{T}{\rightharpoonup} yz \;\wedge\; z \overset{T}{\rightharpoonup} wv \Rightarrow x \overset{T}{\rightharpoonup} yzv \tag{60}$$

In fact:

$$\begin{aligned} & x \overset{T}{\rightharpoonup} yz \;\wedge\; z \overset{T}{\rightharpoonup} wv \\ \Rightarrow \quad & \{ \text{ (A2) } \} \end{aligned}$$

$$
\begin{array}{ll}
 & x \overset{T}{\rightharpoonup} yz \;\wedge\; yz \overset{T}{\rightharpoonup} ywv \\
\Rightarrow & \quad \{\ \text{(A3)}\ \} \\
 & x \overset{T}{\rightharpoonup} yz \;\wedge\; x \overset{T}{\rightharpoonup} ywv \\
\equiv & \quad \{\ \text{(46)}\ \} \\
 & x \overset{T}{\rightharpoonup} yzwv \\
\Rightarrow & \quad \{\ \text{(49)}\ \} \\
 & x \overset{T}{\rightharpoonup} yzv
\end{array}
$$

8.2 Attributes

Database (relational) files are coreflexives on n-dimensional Cartesian products $A_1 \times \cdots \times A_n$. Each projection π_i ($i \in n$) is called an attribute. From $\times$-reflexion $\langle \pi_1, \ldots, \pi_n \rangle = id$ we draw that all attributes together are maximal superkeys: $\pi_1 \cdots \pi_n \simeq id$. In fact, any permutation of this split is an isomorphism (eg. $swap$ for $n = 2$) and therefore a maximal superkey. Wherever f is an arbitrary *split* of attributes we denote by $\overline{f}$ the *split* of the remaining attributes, in any order. The $\overline{f}$ notation only makes sense in the context of $\simeq$-equivalence and obeys the following properties:

$$
\begin{aligned}
f\overline{f} &\simeq id \\
\overline{\overline{f}} &\simeq f
\end{aligned}
$$

9 Multi-valued dependencies

Definition 3. *Given subsets $x, y \subseteq S$ of the relation scheme S of a relation R, this relation is said to satisfy the* multi-valued *dependency (MVD) $x \twoheadrightarrow y$ iff, for any two tuples $t, t' \in R$ which "agree" on x there exists a tuple $t'' \in R$ which "agrees" with t on xy and "agrees" with t' on $S - xy$, that is,*

$$
\begin{array}{l}
\langle \forall\, t, t' \ :\ t, t' \in R : \quad t[x] = t'[x] \\
\qquad\qquad \Downarrow \\
\qquad \langle \exists\, t'' \ :\ t'' \in R :\ t[xy] = t''[xy] \wedge \\
\qquad\qquad t''[S - xy] = t'[S - xy] \ \rangle \quad \rangle
\end{array} \tag{61}
$$

cf. the following picture:

	$S-xy$	x	y
t	γ	α	β
t''	γ'	α	β
t'	γ'	α	β'

□

Our efforts towards writing (61) without variables will be considerably softened by following the rules which follow, one generalizing relational inclusion and the other relational composition:

– Given $B \xleftarrow{R,S} A$ and two predicates $2 \xleftarrow{\psi} A$ and $2 \xleftarrow{\phi} B$ (coreflexively denoted by Ψ and Φ, respectively), then

$$\begin{array}{r}\langle \forall\, b,a \;:\; (\phi\; b)\;\wedge\;(\psi\; a):\; b\,R\,a \Rightarrow b\,S\,a\rangle \equiv\\ \langle \forall\, b,a \;::\; b(\Phi\cdot R\cdot\Psi^\circ \subseteq S)a\rangle\end{array} \tag{62}$$

extends (9), which corresponds to the special case $\Phi = \Psi = id$. (In retrospect, notice this is the rule implicit in the reasoning carried out in section 4.)

– Moreover, for $B \xleftarrow{R} A$ and $A \xleftarrow{S} C$ two binary relations,

$$\begin{array}{r}\langle \forall\, b,c \;::\; \langle \exists\, a \;:\; \psi\; a:\; b\,R\,a \;\wedge\; a\;Sc\rangle\rangle \equiv\\ \langle \forall\, b,c \;::\; b(R\cdot\Psi\cdot S)c\rangle\end{array} \tag{63}$$

extends relational composition (for $\Psi = id$ we are back to $R\cdot S$).

In the spirit of the $\overline{f}$ notation of section 8.2, we denote $S - xy$ by $\overline{xy}$ in the following conversion of the existential quantification in (61) into pointfree notation:

$$\begin{array}{ll} & \langle \exists\, t'' \;:\; t''\in R:\; t[xy] = t''[xy] \;\wedge\; t''[\overline{xy}] = t'[\overline{xy}]\rangle \\ \equiv & \quad \{\; \text{(63) for } \phi := (\in R)\text{, an so on } \} \\ & t(\mathsf{ker}\; xy\cdot [\![R]\!]\cdot \mathsf{ker}\;\overline{xy})t' \end{array}$$

Then we include this in the overall formula and reason:

$$\begin{array}{ll} & \langle \forall\, t,t' \;:\; t,t'\in R:\; t[x] = t'[x] \Rightarrow t(\mathsf{ker}\; xy\cdot [\![R]\!]\cdot \mathsf{ker}\;\overline{xy})t'\,\rangle \\ \equiv & \quad \{\; \text{rule (62) for } \phi = \psi = (\in R) \;\} \\ & \langle \forall\, t,t' \;::\; t(\,[\![R]\!]\cdot(\mathsf{ker}\; x)\cdot[\![R]\!]^\circ \subseteq \mathsf{ker}\; xy\cdot[\![R]\!]\cdot\mathsf{ker}\;\overline{xy})t'\,\rangle \\ \equiv & \quad \{\; \text{kernel of composition } \} \\ & \langle \forall\, t,t' \;::\; t(\,\mathsf{ker}\;(x\cdot[\![R]\!]^\circ) \subseteq \mathsf{ker}\; xy\cdot[\![R]\!]\cdot\mathsf{ker}\;\overline{xy})t'\,\rangle \end{array}$$

Thus we reach the following pointfree definition, in which we generalize $[\![R]\!]$ to an arbitrary endo-relation $A \xleftarrow{R} A$:

$$x \overset{R}{\twoheadrightarrow} y \overset{\text{def}}{=} \mathit{ker}\,(x \cdot R^\circ) \subseteq (\mathit{ker}\, xy) \cdot R \cdot \mathit{ker}\, \overline{xy} \tag{64}$$

Why does definition (64) require R to have the same source and target type? Just expand the right hand-side of (64) to

$$xy \cdot (\mathit{ker}\,(x \cdot R^\circ)) \cdot \overline{xy}^\circ \subseteq xy \cdot R \cdot \overline{xy}^\circ$$

and even further to

$$(xy \cdot R \cdot x^\circ) \cdot (x \cdot R^\circ \cdot \overline{xy}^\circ) \subseteq xy \cdot R \cdot \overline{xy}^\circ$$

and draw diagram

$$\begin{tikzcd}
A \arrow[dr, "x"'] \arrow[dd, "xy"'] & & A \arrow[ll, "R"'] \arrow[dl, "x"] \arrow[dd, "\overline{xy}"] \\
& B \arrow[dl, "xy\cdot R\cdot \overline{x}^\circ"'] \arrow[d, phantom, "\subseteq"] & \\
B \times C & {} & D \arrow[ul, "x\cdot R\cdot \overline{xy}^\circ"'] \arrow[ll, "xy\cdot R\cdot \overline{xy}^\circ"]
\end{tikzcd}$$

Therefore, MVD $x \overset{R}{\twoheadrightarrow} y$ requires R to be an endorelation. This diagram provides and alternative meaning for MVDs: $x \overset{R}{\twoheadrightarrow} y$ holds iff *the projection of R via $xy, \overline{xy}$ "factorizes" through x.*

As happens with FDs, the axiomatic theory of MVDs assumes R to be "a set of tuples". As was have done before, we model such a set by a coreflexive relation as use capital letter T to stress this assumption.

10 Reasoning

MVDs are less intuitive than FDs. In fact, it is known from the standard theory that FDs are just a particular case of MVDs, that is,

$$x \overset{T}{\rightharpoonup} y \Rightarrow x \overset{T}{\twoheadrightarrow} y \tag{65}$$

holds. Our proof of this fact (often termed the *conversion axiom*) is as follows:

$$
\begin{array}{ll}
 & x \overset{T}{\rightharpoonup} y \\
\Rightarrow & \quad \{ \text{ augmentation (53) for } z := x \ \} \\
 & x \overset{T}{\rightharpoonup} xy \\
\equiv & \quad \{ \text{ FD definition (17) } \} \\
 & \mathit{ker}\,(x \cdot T^{\circ}) \subseteq \mathit{ker}\, xy \\
\Rightarrow & \quad \{ \text{ composition is monotone, } T = T^{\circ} = T \cdot T \text{ for coreflexive } T \} \\
 & \mathit{ker}\,(x \cdot T^{\circ}) \subseteq \mathit{ker}\, xy \cdot T \\
\Rightarrow & \quad \{ \text{ in general, } f \leq id, \text{ thus } T \subseteq T \cdot \mathit{ker}\, f \ \} \\
 & \mathit{ker}\,(x \cdot T^{\circ}) \subseteq (\mathit{ker}\, xy) \cdot T \cdot \mathit{ker}\, \overline{xy} \\
\equiv & \quad \{ \text{ definition (64) } \} \\
 & x \overset{T}{\rightharpoonup\!\!\rightharpoonup} y
\end{array}
$$

This axiom is given in [Mai83] as a corollary of the theorem of *lossless decomposition* of MVDs. Instead of resorting to such an indirect proof method, which requires further ingredients of the underlying theory, our proof by calculation is performed in the same style as earlier on for FDs.

MVD theory generalizes FD theory. Some results stem directly from the conversion axiom, as is the case of the MVD reflexivity axiom,

$$y \leq x \Rightarrow x \overset{T}{\rightharpoonup\!\!\rightharpoonup} y \tag{66}$$

since

$$
\begin{array}{ll}
 & x \overset{T}{\rightharpoonup\!\!\rightharpoonup} y \\
\Leftarrow & \quad \{ \text{ conversion (65) } \} \\
 & x \overset{T}{\rightharpoonup} y \\
\Leftarrow & \quad \{ \text{ FD reflexivity (59) } \} \\
 & y \leq x
\end{array}
$$

Some others are new, for instance the *complementation axiom*:

$$x \overset{R}{\rightharpoonup\!\!\rightharpoonup} y \Rightarrow x \overset{R}{\rightharpoonup\!\!\rightharpoonup} \overline{y} \tag{67}$$

which the reader may wish to prove as an exercise. All other MVD inference axioms can be found in [Mai83], section 7.4.1. In this paper we don't go beyond this point.

11 Conclusions

When computer scientists refer to the *relational calculus*, by default what is understood is the calculus of n-ary relations studied in logics and database theory, and not the calculus of *binary* relations which was initiated by De Morgan in the 1860s [Pra92] an eventually became the basis of the *algebra of programming* [BdM97,Bac04].

According to [Kan03], it was Quine, in his 1932 Ph.D. dissertation, who showed how to develop the theory of n-ary relations for all n simultaneously, by defining ordered n-tuples in terms of the ordered pair. (Norbert Wiener is apparently the first mathematician to publicly identify, in the 1910s, n-ary relations with subsets of n-tuples.) Codd discovered and publicized procedures for constructing a set of simple n-ary relations which can support a set of given data and constructed an extension of the calculus of binary relations capable of handling most typical data retrieval problems.

Since binary relations are just n-ary relations, for $n = 2$, there seems to be little point in explaining n-ary relational theory in terms of binary relations. When Codd talks about the binary relation representation of an n-ary relation in [Cod70], one has the feeling that there are more disadvantages than advantages in such a representation.

Contrary to these intuitions, this paper shows that such a strategy makes sense, at least in the database theory context. Classical pointwise relational database theory is full of lengthy formulæ, and proofs with lots of $\cdots$ notation, case analyses and English explanations for "obvious" steps. The adoption of the (pointfree) binary relation calculus is beneficial in several main respects. First, the fact that pointfree notation abstracts from "points" or variables makes the reasoning more compact and effective. Elegant formulæ such as (29) — when compared with (2) — come in support for this claim. Second, proofs are performed by easy-to-follow calculations. Third, one is able to generalize the original theory, as happens with our generalization of attributes to arbitrary (suitably typed) functions in FDs and MVDs.

In retrospect, the use of *coreflexive* relations to model sets of tuples and predicates in the binary relation calculus (instead of arbitrarily partitioning attributes in "source ones" and "target ones") is perhaps the main ingredient of the simplification and subsequent generalization. (A similar strategy has been followed in [OR04] concerning a pointfree model of *hash tables*).

12 Future work

This paper addresses the foundations of FD theory in a pointfree style. No claim is made for extending or improving the standard theory. What is gained is a better starting point for an immense body of knowledge. (FDs are the subject of no more than 20% of the pages Maier's book on relational database theory [Mai83].)

The effectiveness of the approach can only be tested once more and more results are dealt with. In this paper, multivalued dependencies have only been hinted at. Join dependencies have not been considered at all.

Outside the database context, functional dependencies have been used to solve ambiguities in multiple parameter type classes in the Haskell type system [Jon00]. This may happen to be an area of application of the reasoning techniques developed in this paper.

At the level of foundations, *left* and *right conditions* [Hoo97] should be also exploited as alternatives to coreflexives. Concerning representation theory and lossless decomposition, some recent results in [Oli04] and [Rod05] should be taken into account and generalized. This brings us back (albeit in a sketchy way) to our motivation on spreadsheet normalization.

Lossless decomposition. Maintaining arbitrary FDs can be hard, because they constrain the update, insert, delete operations on database files and waste space. Therefore, instead of allowing a relation R to satisfy an arbitrary FD, it is preferable to "extract" such a dependency by decomposing R in two parts — the FD itself, eg.

$$\{\textsc{Pilot}, \textsc{Flight}, \textsc{Date}\}$$

and the "rest" of R, eg.

$$S = \{\textsc{Pilot}, \textsc{Flight}, \textsc{Departs}\}$$

in the example presented in the beginning of this paper.

In general, let $f \overset{R}{\rightharpoonup} g$ be a functional dependency. Then R can be factorized in two components, the functional dependency itself $g \cdot R \cdot f^\circ$ and the "rest" of R, which we model by post-composing R with some selector function h:

$$\begin{tikzcd}
E & \\
A \arrow[u, "h"'] \arrow[d, "g"'] & B \arrow[l, "R"'] \arrow[ul, "h\cdot R"'] \arrow[d, "f"] \\
C & D \arrow[l, "g\cdot R\cdot f^\circ"]
\end{tikzcd}$$

Can we recover R from the two components $h \cdot R$ and $g \cdot R \cdot f^\circ$? The answer will be "yes" provided there is some $E \times C \xrightarrow{j} A$ such that

$$R = j \cdot \langle h \cdot R, g \cdot R \cdot f^\circ \rangle$$

Such is the case of, for instance, $j = id$, $g = \pi_1$ and $h = \pi_2$, or $j = swap$, $g = \pi_2$ and $h = \pi_1$, etc.

Lossless decompositions of this kind provide a representation theory for n-ary relations which isolate FDs and save disk space. FDs and MVDs are just the invariant properties on R which make such decompositions lossless. Thus their relevance in the whole theory.

References

[Bac04] R.C. Backhouse. *Mathematics of Program Construction*. 2004. Draft book in preparation.

[BB03] K. Backhouse and R.C. Backhouse. Safety of abstract interpretations for free,via logical relations and Galois connections. *Science of Computer Programming*, 2003. Accepted for publication.

[BdM97] R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997. C.A. R. Hoare, series editor.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, June 1970.

[GUW02] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002. ISBN: 0-13-031995-3.

[Hoo97] Paul Hoogendijk. *A Generic Theory of Data Types*. PhD thesis, University of Eindhoven, The Netherlands, 1997.

[Jon00] Mark P. Jones. Type classes with functional dependencies. In Gert Smolka, editor, *Programming Languages and Systems, 9th European Symposium on Programming, ESOP 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1782 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2000.

[Kan03] Akihiro Kanamori. The empty set, the singleton, and the ordered pair. *The Bulletin of Symbolic Logic*, 9(3):273–298, 2003.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983. ISBN 0-914894-42-0.

[Oli04] J.N. Oliveira. *Calculate databases with 'simplicity'*, September 2004. Presentation at the *IFIP WG 2.1 #59 Meeting*, Nottingham, UK.

[OR04] J.N. Oliveira and C.J. Rodrigues. Transposing relations: from *Maybe* functions to hash tables. In *MPC'07 : Seventh International Conference on Mathematics of Program Construction, 12-14 July, 2004, Stirling, Scotland, UK (Organized in conjunction with AMAST'04)*, volume 3125 of *Lecture Notes in Computer Science*, pages 334–356. Springer, 2004.

[Pra92] V. Pratt. Origins of the calculus of binary relations. In *Proc. of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 248–254, Santa Cruz, CA, 1992. IEEE Computer Soc.

[Rod05] C.J. Rodrigues. Data refinement by calculation, 2005. Presented at the DI/UM Ph.D. Student Symposium, Quinta da Torre, Soutelo, 5-7 Jan. 2005.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.