First Steps in Pointfree Functional Dependency Theory (DRAFT OF OCTOBER 27, 2005)

J.N. Oliveira

Dep. Informática, Universidade do Minho, Campus de Gualtar, 4700-320 Braga, Portugal,
jno@di.uminho.pt

Abstract. When software designers refer to the *relational calculus*, what they usually mean is the set-theoretic kernel of relational database design "à la Codd" and not the calculus of *binary* relations which was initiated by De Morgan in the 1860s an eventually became the core of the *algebra of programming*.
Contrary to the intuition that a binary relation is just a particular case of n-ary relation, this paper shows the effectiveness of the former in "explaining" and reasoning about the latter. The theory of functional dependencies, which is central to such database design techniques, is addressed in a pointfree style instead of reasoning in the standard set-theoretic model "à la Codd".
It turns out that the theory becomes more general and considerably simpler. Elegant expressions replace lengthy formulæ and easy-to-follow calculations replace pointwise proofs with lots of "$\cdots$" notation, case analyses and natural language explanations for "obvious" steps.

1 Introduction

In standard relational data processing, 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 some Excel spreadsheet — and each column lists the values of a particular attribute in all tuples (eg. row "A" in the same 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, $R \subseteq A_1 \times \ldots \times A_n$. A *relational database* is a collection of several such n-ary relations.

When software designers 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 [15] an eventually became the core of the *algebra of programming* [1, 4, 3].

According to [11], 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.) Since the 1970s, the information system community is indebted to Codd for his pioneering work on the foundations of the *relational data model* theory [5].

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 then, relational database theory has been thoroughly studied, and several textbooks are available on the topic, namely [12], [17] and [6].

The common understanding is that binary relations are just n-ary relations, for $n = 2$, and so there seems to be little point in explaining n-ary relational theory in terms of binary relations. As a matter of fact, when Codd talks about the binary relation representation of an n-ary relation in [5], one has the feeling that there are more disadvantages than advantages in such a representation. Contrary to these intuitions, this paper aims at showing that such a strategy makes sense, at least in *functional dependency* theory, the subset of database theory actually addressed in this paper. (Outside the database context, functional dependencies have been used to solve ambiguities in multiple parameter type classes in the Haskell type system [10].)

Classical pointwise relational database theory is full of lengthy formulæ, and proofs with lots of "..." notation, case analyses and English explanations for "obvious" steps. We show that the adoption of the (pointfree) binary relation calculus is beneficial in several respects. First, the fact that pointfree notation abstracts from "points" or variables makes the reasoning more compact and effective. Second, proofs are performed by easy-to-follow calculations. Third, one is able to generalize the original theory, as will happen with our generalization of attributes to arbitrary (suitably typed) functions in functional dependencies and multi-valued dependencies.

Paper structure. This paper is structured as follows. First we introduce the standard notion of a *functional dependency* (FD) and revise the pointfree theory of functions. Both worlds are combined in section 5, where FDs are presented in the pointfree style. Sections 6 and 7 generalize (pointfree) FD-theory by moving from attribute projections to arbitrary functions. Sections 8 to 10 provide calculational proofs for the standard FD-theory, inc. the Armstrong-axioms and the theorem of lossless decomposition. Multi-valued dependencies are the subject of section 11. The remainder of the paper presents our conclusions and prospect for future work and (in the appendices) some auxiliary results.

2 What is a functional dependency?

In an n-ary relation, attribute *names* normally replace natural numbers in the identification of attributes. The enumeration of all attribute names in a database relation, for instance

$$S = \{\text{PILOT}, \text{FLIGHT}, \text{DATE}, \text{DEPARTS}\} \tag{1}$$

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

[1] This example is taken from [12].

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 *and* DATE. In the standard practice, this will be abbreviated by writing

$$\text{FLIGHT DATE} \rightarrow \text{PILOT}$$

which has the following, alternative reading: FLIGHT *and* DATE *functionally determine* PILOT. Another FD in this example is

$$\text{FLIGHT} \rightarrow \text{DEPARTS} \tag{2}$$

since a given flight always departs at the same time.

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?

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 $x \rightarrow y$ iff all pairs of tuples $t, t' \in R$ which "agree" on x also "agree" on y, that is,*

$$\langle \forall\, t, t' \;:\; t, t' \in R :\; t[x] = t'[x] \Rightarrow t[y] = t'[y] \rangle \tag{3}$$

(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* [12] which can be used as inference rules for FDs. Equivalent axioms have been found which make FD checking more efficient.

Why has this theory "gone this way"? Perhaps one reason lays in the fact that formula (3), 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 paper 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 the qualifier "functional" in "functional dependency" stems from "function", of course. So our first effort goes into making sure we have a clear idea of "what a function is".

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

A function f is a special case of binary relation satisfying two main properties:

– *"Left" Uniqueness*

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

– *Leibniz principle*

$$a = a' \Rightarrow f\ a = f\ a' \tag{5}$$

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:

$$\mathsf{img}\ f \subseteq id \tag{6}$$

– f is entire:

$$id \subseteq \mathsf{ker}\ f \tag{7}$$

Formulæ (6) and (7) are examples of *pointfree* notation in which *points* — eg. a, a', b, b' in (4,5) — 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 such as $\mathsf{ker}\ f$ (read: "*kernel* of f") and $\mathsf{img}\ f$ (read: "*image* of f),

$$\mathsf{ker}\ R = R^\circ \cdot R \tag{8}$$

$$\mathsf{img}\ R = R \cdot R^\circ \tag{9}$$

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 $A \xleftarrow{S} C$ are two binary relations on datatypes A (source) and B (target) and C (source) and A (target), respectively. Converse commutes with composition in a contravariant way,

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{10}$$

and so image and converse commute via converse:

$$\mathsf{ker}\ (R^\circ) = \mathsf{img}\ R \tag{11}$$

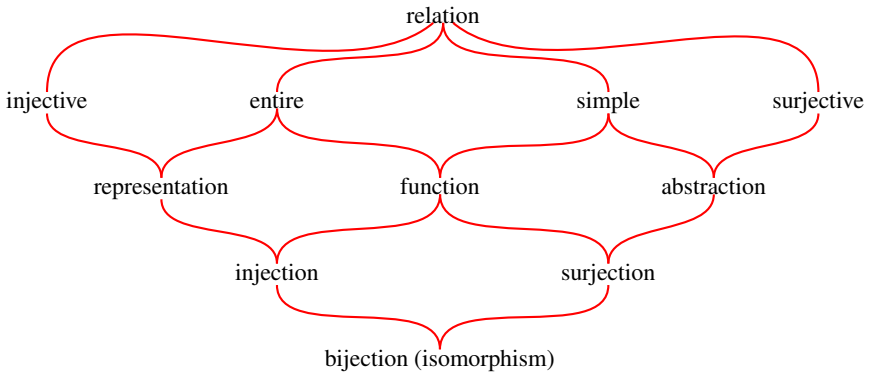
$$\mathsf{img}\ (R^\circ) = \mathsf{ker}\ R \tag{12}$$

As in (6) and (7), the underlying partial order on relations is written $R \subseteq S$, meaning

$$R \subseteq S \equiv \langle \forall\ b, a\ ::\ bRa \Rightarrow bSa \rangle \tag{13}$$

for all a and b suitably typed.

The *simple* and *entire* classes of relation mentioned above are 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

(14)

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 relation $[\![p]\!]$ 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 the meaning of its *characteristic* predicate $[\![\lambda a.a \in S]\!]$, that is,

$$b[\![S]\!]a \equiv (b = a) \wedge a \in S \tag{15}$$

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

Before we embark on converting (3) into pointfree notation, let us see an alternative view of functions better suited for calculation.

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

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

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S \tag{16}$$

$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f \tag{17}$$

As a warming-up exercise, let us check one of these, say (16). (The whole picture can be found in eg. [8, 4, 3].) That f entire and simple *implies* equivalence (16) can be proved by mutual implication:

$$
\begin{array}{ll}
& f \cdot R \subseteq S \\
\Rightarrow & \quad \{ \text{ monotonicity of composition } \} \\
& f^\circ \cdot f \cdot R \subseteq f^\circ \cdot S \\
\Rightarrow & \quad \{ \ f \text{ is entire (7) } \} \\
& R \subseteq f^\circ \cdot S \\
\Rightarrow & \quad \{ \text{ monotonicity of composition } \} \\
& f \cdot R \subseteq f \cdot f^\circ \cdot S \\
\Rightarrow & \quad \{ \ f \text{ is simple (6) } \} \\
& f \cdot R \subseteq S
\end{array}
$$

That (16) *implies* that f is entire and simple can be checked by instantiating $R, S := id, f$ (left-cancellation) or $S, R := id, f^\circ$ (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 [2] :

$$b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a) \tag{18}$$

The use of (18) to convert (4, 5) into (6, 7), respectively, is left as an exercise.

5 FD-satisfiability in pointfree style

Attributes are functions. Let R be a n-ary relation with schema S, t be a tuple in R and a be an attribute in S. Notation $t[a]$ was adopted in (3) 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 (3) 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:

$$
\begin{array}{ll}
& t \in R \ \wedge\ t' \in R \ \wedge\ (x\ t) = (x\ t') \Rightarrow (y\ t) = (y\ t') \\
\equiv & \quad \{ \ (18) \text{ twice, for } R := id \ \} \\
& t \in R \ \wedge\ t' \in R \ \wedge\ t(x^\circ \cdot x\ )t' \Rightarrow t(y^\circ \cdot y\ )t' \\
\equiv & \quad \{ \ (15) \text{ twice } \} \\
& t[\![R]\!]u \ \wedge\ t = u \ \wedge\ t'[\![R]\!]u' \ \wedge\ t' = u' \ \wedge\ t(x^\circ \cdot x\ )t' \Rightarrow t(y^\circ \cdot y\ )t'
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{\ \wedge \text{ is commutative; substitution of equals for equals; converse}\ \} \\
& t[\![R]\!]u \ \wedge\ u(x^{\circ}\cdot x)u' \ \wedge\ u'[\![R]\!]^{\circ}t' \Rightarrow t(y^{\circ}\cdot y)t' \\
\equiv & \{\ \text{composition ; relation inclusion (13)}\ \} \\
& [\![R]\!]\cdot(x^{\circ}\cdot x)\cdot[\![R]\!]^{\circ} \subseteq y^{\circ}\cdot y \qquad (19) \\
\equiv & \{\ \text{shunting rules (16) and (17)}\ \} \\
& y\ \cdot[\![R]\!]\cdot x^{\circ}\cdot x\ \cdot[\![R]\!]^{\circ}\cdot y^{\circ} \subseteq id \\
\equiv & \{\ \text{converse versus composition, } (R\cdot S)^{\circ} = S^{\circ}\cdot R^{\circ}\text{, followed by (9)}\ \} \\
& \mathsf{img}\,(y\ \cdot[\![R]\!]\cdot x^{\circ}) \subseteq id
\end{array}
$$

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

$$y\ \cdot[\![R]\!]\cdot x^{\circ} \qquad (20)$$

is simple, cf. (6).

6 Functional dependencies in general

Definition. Our own definition of FD starts from the observation that coreflexive relation $[\![R]\!]$ and projection functions x and y in (20) 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*

$$\begin{array}{ccc} A & \xleftarrow{R} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle f} \\ C & \xleftarrow[g\cdot R\cdot f^{\circ}]{} & D \end{array}$$

is simple. Equivalent definitions are

$$f \overset{R}{\rightharpoonup} g \equiv\ R\cdot(\mathsf{ker}\,f)\cdot R^{\circ} \subseteq \mathsf{ker}\,g \qquad (21)$$

— cf. (19) and (8) — and

$$f \overset{R}{\rightharpoonup} g \equiv\ \mathsf{ker}(f\cdot R^{\circ}) \subseteq \mathsf{ker}\,g \qquad (22)$$

thanks to (10).

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

Generic relational projection. The little piece of notation which follows will be of some help in the sequel: given a binary relation R and functions f and g such as in definition 2, we define the f, g*-projection* of R as binary relation

$$\pi_{g,f} R \stackrel{\text{def}}{=} g \cdot R \cdot f^\circ \tag{23}$$

So, fact $f \stackrel{R}{\perp} g$ in definition 2 can be rephrased by saying that projection $\pi_{g,f} R$ is simple.

It can be shown that definition (23) extends that of the standard n-ary relation *project* operator, whose set-theoretic semantics are as follows [12], for n-ary relation T with schema S and $X \subseteq S$:

$$\pi_x T = \{x[t] \mid t \in T\} \tag{24}$$

In fact, while combining the lower adjoints of shunting rules (16, 17), $\pi_{g,f}$ is itself a lower adjoint,

$$\pi_{g,f} R \subseteq S \equiv R \subseteq g^\circ \cdot S \cdot f \tag{25}$$

meaning that $\pi_{g,f} R$ is the *smallest* relation S such that, wherever a is R-related to b, $(g\ a)$ is S-related to $(f\ b)$ — recall (18). Regarding R and S as sets of pairs, we have

$$\pi_{g,f} R \stackrel{\text{def}}{=} \{(g\ a, f\ b) \mid (a, b) \in R\}$$

It can be easily shown that $\pi_{f,f} R$ is coreflexive wherever R is coreflexive. From this we draw, for n-ary relation T such as in (24):

$$[\![\pi_x T]\!] = \pi_{x,x} [\![T]\!]$$

(Note the use of the same symbol π to denote both the standard set-theoretic projection operator and the pointfree one.)

Besides monotonicity and $\cup$-preservation, ensured by lower-adjointedness, binary relation projection obeys to a number of useful properties, namely:

$$\pi_{id,id} = id \tag{26}$$

$$\pi_{f,g} \cdot \pi_{h,k} = \pi_{f \cdot h, k \cdot g} \tag{27}$$

$$(\pi_{f,g} R)^\circ = \pi_{g,f} (R^\circ) \tag{28}$$

$$\langle \pi_{f,g}, \pi_{h,g} \rangle = \pi_{\langle f,h \rangle, g} \tag{29}$$

Another, quite interesting view of (25) is

$$\pi_{g,f} R \subseteq S \equiv g (S \leftarrow R) f \tag{30}$$

where $S \leftarrow R$ is Reynolds "arrow combinator"

$$g (S \leftarrow R) f \equiv g \cdot R \subseteq S \cdot f \tag{31}$$

which is extensively studied in [2]. So, a (f, g-parametric) relation between two relations (R and S) can be equated as a (R, S-parametric) relation on the projection functions f and g themselves.

Examples. The reader may wish to check that $f \overset{R}{\leftarrow} g$ 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 (3), equations (21) and (22) 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:

$$f \overset{\perp}{\leftarrow} g \tag{32}$$

(where $\perp$ denotes the empty relation)

$$f \overset{R}{\leftarrow} ! \tag{33}$$

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

$$id \overset{R}{\leftarrow} id \equiv \ R \text{ is simple} \tag{34}$$

$$f \overset{R}{\leftarrow} f \Leftarrow R \subseteq id \tag{35}$$

An immediate consequence of (35) is

$$f \overset{id}{\leftarrow} f \tag{36}$$

7 The role of injectivity

Ordering relations by injectivity. It can be observed that what matters about f and g in (21) is their "degree of injectivity" as measured by $\mathsf{ker}\, f$ and $\mathsf{ker}\, g$, in opposite directions: more injective f and less injective g will strengthen a given FD $f \overset{R}{\leftarrow} g$. An extreme case is $f = id$ and $g = !$ — functional dependency $id \overset{R}{\leftarrow} !$ will always hold for any R, cf. (33).

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

$$R \leq S \equiv \mathsf{ker}\, S \subseteq \mathsf{ker}\, R \tag{37}$$

that is, $R \leq S$ means R is *less* injective than S [2]. 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

[2] To be more precise, we should write *"less injective or more defined"* since ker measures both properties, recall (14). In case of functions, $f \leq g$ unambiguously means that f is less injective than g.

that

$$! \leq R \tag{38}$$
$$R \leq \bot \tag{39}$$

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

The fact pre-composition respects the injectivity preorder,

$$R \leq S \Rightarrow R \cdot T \leq S \cdot T \tag{40}$$

is easy to prove:

$$\begin{array}{ll} & R \leq S \\ \equiv & \quad \{ \text{ (37) and (8) } \} \\ & S^\circ \cdot S \subseteq R^\circ \cdot R \\ \Rightarrow & \quad \{ \text{ monotonicity of } (T^\circ \cdot \ ) \text{ and } (\ \cdot T) \ \} \\ & T^\circ \cdot S^\circ \cdot S \cdot T \subseteq T^\circ \cdot R^\circ \cdot R \cdot T \\ \equiv & \quad \{ \text{ (10) twice, followed by (8,37) } \} \\ & R \cdot T \leq S \cdot T \end{array}$$

(This proof instantiates a more general construction presented in appendix A.)

FD defined via the injectivity ordering. The close relationship between FDs and injectivity of observations is well captured by the following re-statement of (22) in terms of (37):

$$f \xrightarrow{R} g \ \equiv \ g \leq f \cdot R^\circ \tag{41}$$

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,

$$f \xrightarrow{S \cdot R} h \Leftarrow f \xrightarrow{R} g \ \wedge \ g \xrightarrow{S} h \tag{42}$$

which follows:

$$\begin{array}{ll} & f \xrightarrow{R} g \ \wedge \ g \xrightarrow{S} h \\ \equiv & \quad \{ \text{ (41) twice } \} \\ & g \leq f \cdot R^\circ \ \wedge \ h \leq g \cdot S^\circ \\ \Rightarrow & \quad \{ \leq\text{-monotonicity of } (\ \cdot S^\circ) \text{ (40) followed by (10) } \} \\ & g \cdot S^\circ \leq f \cdot (S \cdot R)^\circ \ \wedge \ h \leq g \cdot S^\circ \end{array}$$

$$
\begin{array}{ll}
\Rightarrow & \{\ \leq\text{-transitivity}\ \} \\
& h \leq f \cdot (S \cdot R)^\circ \\
\equiv & \{\ (41)\ \text{again}\ \} \\
& f \stackrel{S \cdot R}{\leftarrow} h
\end{array}
$$

A category of functions. Note in passing that (42) and (36) 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 $f \stackrel{R}{\leftarrow} g$.

Simultaneous observations. In the same way x and y in (3) may involve more that one observable attribute, we would like f and g in (21) to involve more than one *observation* function. Multiple observations add more detail and so are likely to be more injective. The relational *split* combinator

$$(a, b)\langle R, S\rangle c \equiv a\ R\ c\ \wedge\ b\ S\ c \tag{43}$$

captures this effect, and facts

$$R \leq \langle R, S\rangle \text{ and } S \leq \langle R, S\rangle \tag{44}$$

are easy to check by recalling

$$\mathsf{ker}\,\langle R, S\rangle = (\mathsf{ker}\,R) \cap (\mathsf{ker}\,S) \tag{45}$$

which stems from

$$\langle R, S\rangle^\circ \cdot \langle X, Y\rangle = (R^\circ \cdot X) \cap (S^\circ \cdot Y) \tag{46}$$

cf. [4]. Moreover, the following Galois connection

$$\langle R, S\rangle \leq T \equiv R \leq T\ \wedge\ S \leq T \tag{47}$$

stems from the one underlying $\cap$, as shown in appendix A. The anti-symmetric closure of $\leq$ yields an equivalence relation

$$R \simeq S \equiv \mathsf{ker}\,R = \mathsf{ker}\,S \tag{48}$$

which is such that, for instance, $! \simeq \top$ holds. We also have

$$S \leq R \equiv \langle S, R\rangle \simeq R \tag{49}$$

The following equivalences will be relevant in the sequel, for suitably typed R, S and T:

$$R \simeq \langle R, R\rangle \tag{50}$$

$$\langle R, S\rangle \simeq \langle S, R\rangle \tag{51}$$

$$\langle T, \langle R, S\rangle\rangle \simeq \langle\langle T, R\rangle, S\rangle \tag{52}$$

Function injectivity. Since attributes in the n-ary relational database model are (projection) 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,

$$! \leq f \leq id \tag{53}$$

and

$$f \simeq id \equiv \ f \text{ is an injection} \tag{54}$$

From (53) and (40) we obtain $f \cdot R \leq R$. From (6) we draw $id \leq f^\circ$ and thus $S \leq f^\circ \cdot S$, thanks to (40).

More generally, Galois connection

$$R \cdot g \leq S \equiv R \leq S \cdot g^\circ \tag{55}$$

holds — cf. proof in appendix A — which can be regarded as the "injectivity counterpart" of "shunting" rules (16,17).

FDs on functions. As special cases of relations, functions may also satisfy functional dependencies. For instance, it will be easy to show that $bagify \stackrel{setify}{\rightarrow} id$ holds, where $bagify$ (resp. $setify$) is the function which extracts, from a finite list, the bag (resp. set) of all its elements. From (55) we draw:

$$g \cdot h \leq f \equiv f \stackrel{h}{\rightarrow} g \equiv g \leq f \cdot h^\circ \tag{56}$$

Thus the equivalence

$$g \leq f \equiv f \stackrel{id}{\rightarrow} g \tag{57}$$

(let $h = id$) and a more general pattern of FD chaining

$$f \stackrel{S \cdot R}{\rightarrow} h \Leftarrow f \stackrel{R}{\rightarrow} g \ \wedge \ g \leq j \ \wedge \ j \stackrel{S}{\rightarrow} h \tag{58}$$

which extends (42) via (57).

On $\simeq$-equivalence. The discrimination of functions beyond $\simeq$-equivalence is unnecessary in the context of FD reasoning. Since ordering and repetition in "splits" are $\simeq$-irrelevant — recall (50), (51) and (52) — we will abbreviate $\langle f, g \rangle$ by fg, or by gf, wherever this notation shorthand is welcome and makes sense [3]. Such is the case of a fact which will prove particularly useful in the sequel:

$$f \stackrel{R}{\rightarrow} gh \equiv f \stackrel{R}{\rightarrow} g \ \wedge \ f \stackrel{R}{\rightarrow} h \tag{59}$$

[3] This is inspired by a similar shorthand popular in the standard notation of relational database theory: attribute set union, eg. $X \cup Y$, is denoted by simple juxtaposition, XY.

The proof of (59) is as follows:

$$
\begin{array}{ll}
& f \stackrel{R}{\rightarrow} gh \\
\equiv & \quad \{ \text{ (41) ; expansion of shorthand } gh \ \} \\
& \langle g, h \rangle \leq f \cdot R^{\circ} \\
\equiv & \quad \{ \text{ (47) } \} \\
& g \leq f \cdot R^{\circ} \ \wedge \ h \leq f \cdot R^{\circ} \\
\equiv & \quad \{ \text{ (41) twice } \} \\
& f \stackrel{R}{\rightarrow} g \ \wedge \ f \stackrel{R}{\rightarrow} h
\end{array}
$$

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,

$$h \stackrel{R}{\rightarrow} k \Leftarrow h \geq f \ \wedge \ f \stackrel{R}{\rightarrow} g \ \wedge \ g \geq k \tag{60}$$

and justified:

$$
\begin{array}{ll}
& h \geq f \ \wedge \ f \stackrel{R}{\rightarrow} g \ \wedge \ g \geq k \\
\equiv & \quad \{ \text{ (57) twice } \} \\
& h \stackrel{id}{\rightarrow} f \ \wedge \ f \stackrel{R}{\rightarrow} g \ \wedge \ g \stackrel{id}{\rightarrow} k \\
\Rightarrow & \quad \{ \text{ (42) twice; identity of composition } \} \\
& h \stackrel{R}{\rightarrow} k
\end{array}
$$

The following are corollaries of (60), since $fh \geq f$:

$$fh \stackrel{R}{\rightarrow} g \Leftarrow f \stackrel{R}{\rightarrow} g \tag{61}$$

$$f \stackrel{R}{\rightarrow} g \Leftarrow f \stackrel{R}{\rightarrow} gh \tag{62}$$

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:

$$fh \stackrel{R}{\rightarrow} g \Leftarrow f \stackrel{R}{\rightarrow} gh \tag{63}$$

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

$$fhg \stackrel{R}{\rightarrow} ! \Leftarrow fh \stackrel{R}{\rightarrow} g$$

8 The Armstrong-axioms

In this section we prove the correctness of the Armstrong-axioms [12], 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 sets of attributes of a relational schema S ordered by inclusion, eg. $X \subseteq Y \subseteq S$. Unions of such attribute sets 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 function.)

The whole schema S corresponds to a maximal observation. In our setting, this is captured by the identity function 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 9.)

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:

– F1. **Reflexivity** :

$$x \xrightarrow{T} x \tag{64}$$

— recall (35). Another way to put it is

$$y \leq x \Rightarrow x \xrightarrow{T} y \tag{65}$$

which follows from $x \xrightarrow{T} x \ \wedge \ x \geq y$, recall (60). Another way to express (65) is

$$yz \xrightarrow{T} y \tag{66}$$

— let $x := yz$.

– F2. **Augmentation** :

$$x \xrightarrow{T} y \Rightarrow xz \xrightarrow{T} yz \tag{67}$$

Proof:

$$\begin{array}{ll} & xz \xrightarrow{T} yz \\ \equiv & \quad \{ \ (59) \ \} \\ & xz \xrightarrow{T} y \ \wedge \ xz \xrightarrow{T} z \\ \equiv & \quad \{ \ \text{reflexivity (F1) in version (66)} \ \} \\ & xz \xrightarrow{T} y \\ \Leftarrow & \quad \{ \ (61) \ \} \\ & x \xrightarrow{T} y \end{array}$$

We observe that Maier's version of this axiom is the implication step just above [12].

– F3. **Additivity** (or **Union**):

$$x \xrightarrow{T} y \ \wedge \ x \xrightarrow{T} z \Rightarrow x \xrightarrow{T} yz \tag{68}$$

This is one of the "ping-pong" sides of (59).

– F4. **Projectivity**:

$$x \xrightarrow{T} yz \Rightarrow x \xrightarrow{T} y \ \wedge \ x \xrightarrow{T} z \tag{69}$$

This is the other side of (59).

– F5. **Transitivity** :

$$x \xrightarrow{T} y \ \wedge \ y \xrightarrow{T} z \Rightarrow x \xrightarrow{T} z \tag{70}$$

This stems from (42) for S and R the same coreflexive T, in which case $T \cdot T = T$.

– F6. **Pseudo-transitivity** :

$$x \xrightarrow{T} y \ \wedge \ wy \xrightarrow{T} z \Rightarrow xw \xrightarrow{T} z \tag{71}$$

cf.

$$\begin{array}{ll} & x \xrightarrow{T} y \ \wedge \ wy \xrightarrow{T} z \\ \Rightarrow & \quad \{ \text{ augmentation (F2) } \} \\ & xw \xrightarrow{T} yw \ \wedge \ wy \xrightarrow{T} z \\ \Rightarrow & \quad \{ \text{ transitivity (F5) } \} \\ & xw \xrightarrow{T} z \end{array}$$

This completes the six *inference axioms* which are presented and proved in [12] either directly — using (3) — or indirectly, using tuple counting and properties of two standard n-ary relation operators: *select* and *project*. Our proofs are substantially simpler thanks to the economy of (41) and derived results.

To complete the set, we present below two consequences of the standard axioms which are adopted for efficiency in FD reasoning:

– **Decomposition** :

$$x \xrightarrow{T} y \ \wedge \ z \leq y \Rightarrow x \xrightarrow{T} z \tag{72}$$

This is (60) for $f = k$. Alternatively,

$$\begin{array}{ll} & x \xrightarrow{T} y \ \wedge \ z \leq y \\ \Rightarrow & \quad \{ \text{ (F1) } \} \\ & x \xrightarrow{T} y \ \wedge \ y \xrightarrow{T} z \\ \Rightarrow & \quad \{ \text{ (F5) } \} \\ & x \xrightarrow{T} z \end{array}$$

– **Accumulation** :

$$x \stackrel{T}{\rightarrow} yz \ \wedge \ z \stackrel{T}{\rightarrow} wv \Rightarrow x \stackrel{T}{\rightarrow} yzv \tag{73}$$

In fact:

$$\begin{array}{ll} & x \stackrel{T}{\rightarrow} yz \ \wedge \ z \stackrel{T}{\rightarrow} wv \\ \Rightarrow & \quad \{ \ \text{(F2)} \ \} \\ & x \stackrel{T}{\rightarrow} yz \ \wedge \ yz \stackrel{T}{\rightarrow} ywv \\ \Rightarrow & \quad \{ \ \text{(F5)} \ \} \\ & x \stackrel{T}{\rightarrow} yz \ \wedge \ x \stackrel{T}{\rightarrow} ywv \\ \equiv & \quad \{ \ (59) \ \} \\ & x \stackrel{T}{\rightarrow} yzwv \\ \Rightarrow & \quad \{ \ (62) \ \} \\ & x \stackrel{T}{\rightarrow} yzv \end{array}$$

9 Keys and attributes

Keys. Every x such that $x \stackrel{R}{\rightarrow} id$ — 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 $y \leq x$ such that $y \not\simeq x$, $y \not\stackrel{R}{\rightarrow} id$. In other words,

$$x \text{ is a key of } R \equiv x \stackrel{R}{\rightarrow} id \ \wedge \ \langle \forall \, y \ : \ y \stackrel{R}{\rightarrow} id : \ y \simeq x \vee y \not\leq x \rangle$$

From (34) and (53) we draw that id is always a (maximal) superkey for simple relations.

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. $swap = \pi_2 \pi_1$, 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:

$$\begin{array}{c} f\overline{f} \simeq id \\ \overline{\overline{f}} \simeq f \end{array}$$

10 Lossless decomposition

Arbitrary FDs are, in general, hard to maintain because they constrain the update, insert and delete operations on database files, and waste space. Therefore, instead of allowing some relation T to satisfy an arbitrary FD, it is preferable to "extract" such a dependency by decomposing T in two parts — the FD itself, eg. with schema

$$S_1 = \{\text{FLIGHT}, \text{DEPARTS}\}$$

— recall FD (2) in our introductory example — and the "rest" of T, with schema

$$S_2 = \{\text{PILOT}, \text{FLIGHT}, \text{DATE}\}$$

Such components are referred to, in the standard terminology, as *projections* of T and are denoted by $\pi_{S_1} T$ and $\pi_{S_2} T$, respectively. (Read $\pi_S T$ as *the projection of* T *along schema* S.)

In this example, the fact that FLIGHT — the antecedent of the selected FD — is kept in schema S_2 has to do with the principle of *lossless decomposition*: once T is decomposed in projections $\pi_{S_1} T$ and $\pi_{S_2} T$, by "joining" them one should be able to recover the original relation [4]:

$$(\pi_{S_1} T) \bowtie (\pi_{S_2} T) = T$$

Lossless decomposition is a representation technique which is central to relational database implementation. Of course, not every pair of projections is lossless. A kernel topic of this theory of database design *by decomposition* is precisely that of finding conditions for safe decomposition. Such is the case of extracting functional dependencies, such as illustrated above, thanks to a couple of theorems which will be dealt with in the sequel.

The first of these — exercise 6.4 in [12] — is as follows: *given relation schemes* Y *and* Z *such that* $Y \cap Z = X$ *and a relation* T *with schema* YZ *satisfying FD* $X \to Y$*, then lossless decomposition*

$$T = (\pi_Y T) \bowtie (\pi_Z T)$$

holds.

Our proof of this result boils down to almost *no-work-at-all* thanks to the following binary relation extension of the projection operator given by (23). Recall that (23) expresses the standard semantics of relational projection, the only difference being that (23) requires two projection functions — antecedent f and consequent g — instead

[4] The standard, set-theoretic semantics of the n-ary relation join operator is as follows [12], for relations T, T' with schemes S, S', respectively:

$$T \bowtie T' = \{t \mid \langle \exists\, t, t' \,:\, t \in T \,\wedge\, t' \in T' :\, t = S[t] \,\wedge\, t' = S'[t] \rangle\}$$

of one. This pair leads to a straightforward definition of *join*: joining two projections which share the same antecedent function, say x, is nothing but binary relation *split*:

$$(\pi_{y,x} R) \bowtie (\pi_{z,x} R) \stackrel{\text{def}}{=} \langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle$$

And lossless decomposition can be expressed parametrically with respect to consequent functions y and z,

$$(\pi_{y,x} R) \bowtie (\pi_{z,x} R) = \pi_{yz,x} R$$

that is,

$$\langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle = \langle y, z \rangle \cdot R \cdot x^\circ$$

It is well-known that such unconditioned $\times$-fusion doesn't hold in relation algebra, in general. A theorem in [1] (Theorem 12.30) includes the following side-condition for such a fusion to take place, where R, S, T are suitably typed binary relations:

$$\langle R, S \rangle \cdot T = \langle R \cdot T, S \cdot T \rangle \Leftarrow R \cdot (\mathsf{img}\, T) \subseteq R \vee S \cdot (\mathsf{img}\, T) \subseteq S \qquad (74)$$

For instance, fusion takes place wherever T is simple [5] or wherever S (or R) is a function and T is its converse, eg.

$$\langle R, f \rangle \cdot f^\circ = \langle R \cdot f^\circ, f \cdot f^\circ \rangle \qquad (75)$$

In our case, from (74) we draw $(R, S := y, z)$

$$\langle y, z \rangle \cdot T = \langle y \cdot T, z \cdot T \rangle \Leftarrow y \leq T^\circ \vee z \leq T^\circ$$

— recall (11) and (37) — and, further instantiating $T := R \cdot x^\circ$, we obtain

$$\langle y, z \rangle \cdot (R \cdot x^\circ) = \langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle \quad \Leftarrow \quad x \stackrel{R}{\rightarrow} y \vee x \stackrel{R}{\rightarrow} z$$

In summary, we can establish lossless decomposition via FD extraction as follows, back to the project/join notation:

$$(\pi_{y,x} R) \bowtie (\pi_{z,x} R) = \pi_{yz,x} R \quad \Leftarrow \quad x \stackrel{R}{\rightarrow} y \vee x \stackrel{R}{\rightarrow} z \qquad (76)$$

The question arises: are there side-conditions weaker than that of (76) for lossless decomposition to take place? It turns out that FD existence is a sufficient but not necessary condition for safe decomposition: the more general concept of a *multi-valued* dependency, addressed in the sequel, is what is actually required.

[5] Cf. also [4].

11 Multi-valued dependencies

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

$$\langle \forall\, t, t' \,:\, t, t' \in T : \quad t[x] = t'[x] \quad \rangle \qquad (77)$$
$$\Downarrow$$
$$\langle \exists\, t'' \,:\, t'' \in T : \; t[xy] = t''[xy] \wedge \; t''[S - xy] = t'[S - xy] \rangle$$

cf. the following picture:

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

□

Our efforts towards writing (77) without variables will be considerably softened by the rules which follow, one generalizing relational inclusion and the other relational composition:

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

$$\langle \forall\, b, a \,:\, (\phi\, b) \;\wedge\; (\psi\, a) : \; b\, R\, a \Rightarrow b\, S\, a \rangle \equiv \Phi \cdot R \cdot \Psi^{\circ} \subseteq S \qquad (78)$$

  extends (13), which corresponds to the special case $\Phi = \Psi = id$. (In retrospect, notice this is the rule implicit in the reasoning carried out in section 5.)

- Given two binary relations $B \xleftarrow{R} A$ and $A \xleftarrow{S} C$ and predicate $2 \xleftarrow{\phi} A$ we have, for all b, c

$$\langle \exists\, a \,:\, \phi\, a : \; b\, R\, a \;\wedge\; a\, S c \rangle \equiv b(R \cdot \Phi \cdot S)c \qquad (79)$$

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

In the spirit of the $\overline{f}$ notation of section 9, we denote $S - xy$ by $\overline{xy}$ in the following conversion of the existential quantification of (77) into pointfree notation:

$$\begin{array}{ll} & \langle \exists\, t'' \,:\, t'' \in T : \; t[xy] = t''[xy] \;\wedge\; t''[\overline{xy}] = t'[\overline{xy}] \rangle \\ \equiv & \quad \{ \text{ (79) for } \phi := (\in T), \text{ an so on } \} \\ & t(\ker xy \cdot [\![T]\!] \cdot \ker \overline{xy})t' \end{array}$$

Then we include this in the overall formula and reason:

$$
\begin{array}{ll}
& \langle \forall\, t, t' \;:\; t, t' \in T \;:\; t[x] = t'[x] \Rightarrow t(\ker xy \cdot [\![T]\!] \cdot \ker \overline{xy}) t' \,\rangle \\
\equiv & \quad \{ \text{ rule (78) for } \phi = \psi = (\in T) \ \} \\
& [\![T]\!] \cdot (\ker x) \cdot [\![T]\!]^\circ \subseteq \ker xy \cdot [\![T]\!] \cdot \ker \overline{xy} \\
\equiv & \quad \{ \text{ kernel of composition } \} \\
& \ker (x \cdot [\![T]\!]^\circ) \subseteq \ker xy \cdot [\![T]\!] \cdot \ker \overline{xy}
\end{array}
$$

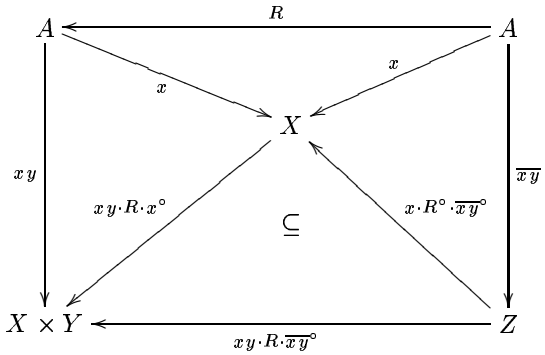
Thus we reach the following pointfree definition, in which we generalize $[\![T]\!]$ to an arbitrary endo-relation $A \xleftarrow{R} A$ and introduce notation $x \overset{R}{\rightharpoonup} y$ (read: *x multi-determines y in R*) in the spirit of $x \overset{R}{\perp} y$ earlier on:

$$x \overset{R}{\rightharpoonup} y \overset{\text{def}}{=} \ker (x \cdot R^\circ) \subseteq (\ker xy) \cdot R \cdot \ker \overline{xy} \tag{80}$$

Why does definition (80) require R to have the same source and target type? Just expand the right hand-side of (80) and "shunt" wherever possible,

$$(xy \cdot R \cdot x^\circ) \cdot (x \cdot R^\circ \cdot \overline{xy}^\circ) \subseteq xy \cdot R \cdot \overline{xy}^\circ \tag{81}$$

to obtain diagram



Therefore, MVD $x \overset{R}{\rightharpoonup} y$ requires R to be an endo-relation. This diagram provides an alternative meaning for MVDs: $x \overset{R}{\rightharpoonup} y$ holds iff projection $\pi_{xy,\overline{xy}} R$ "factorizes" through x. A trivial example of such R is $\bot$, which satisfies *any* MVD. In case of transitive R — ie. such that $R \cdot R \subseteq R$ holds — it is easy to see that condition

$$(\ker x) \cdot R^\circ \subseteq R$$

is sufficient for (81) to hold, since $(xy \cdot R \cdot \_ \cdot \overline{xy})$ is monotonic. Thus $\top$ satisfies *any* MVD.

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

MVDs are more general and less intuitive than FDs. It is known from the standard theory that FDs are just a particular case of MVDs, that is,

$$x \xrightarrow{T} y \Rightarrow x \xrightarrow{T} y \tag{82}$$

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

$$\begin{array}{ll}
& x \xrightarrow{T} y \\
\Rightarrow & \quad \{ \text{ augmentation (67) for } z := x \ \} \\
& x \xrightarrow{T} xy \\
\equiv & \quad \{ \text{ FD definition (22) } \} \\
& \ker(x \cdot T^\circ) \subseteq \ker xy \\
\Rightarrow & \quad \{ \text{ composition is monotone, } T = T^\circ = T \cdot T \text{ for coreflexive } T \ \} \\
& \ker(x \cdot T^\circ) \subseteq \ker xy \cdot T \\
\Rightarrow & \quad \{ \text{ in general, } f \leq id, \text{ thus } T \subseteq T \cdot \ker f \ \} \\
& \ker(x \cdot T^\circ) \subseteq (\ker xy) \cdot T \cdot \ker \overline{xy} \\
\equiv & \quad \{ \text{ definition (80) } \} \\
& x \xrightarrow{T} y
\end{array}$$

Lossless decomposition. The conversion axiom is given in [12] as a corollary of the theorem of *lossless decomposition* of MVDs. This theorem (number 7.1 in [12]) states that fact $x \xrightarrow{T} y$ holds *if and only if* T decomposes losslessly into two relations with schemata xy and $x\overline{yx}$, respectively:

$$x \xrightarrow{T} y \quad \equiv \quad (\pi_{y,x} T) \bowtie (\pi_{\overline{yx},x} T) = \pi_{y\overline{yx},x} T \tag{83}$$

A pointwise proof of this result is given in [12] in "implication-first" logic style, in two parts — the *if* side followed by the *only if* side of the equivalence. Being performed as they are directly over (77), these proofs aren't easy to follow with their existential and universal quantifications over no less than six tuple variables $t, t_1, t_2, t'_1, t'_2, t_3$. By contrast, our proof is a sequence of pointfree equivalences:

$$\begin{array}{ll}
& (\pi_{y,x} T) \bowtie (\pi_{\overline{yx},x} T) = \pi_{y\overline{yx},x} T \\
\equiv & \quad \{ \ (23) \ \} \\
& \langle y \cdot T \cdot x^\circ, \overline{yx} \cdot T \cdot x^\circ \rangle = y\overline{yx} \cdot T \cdot x^\circ
\end{array}$$

$$
\begin{array}{ll}
\equiv & \{\ \text{since } \langle R, S\rangle \cdot T \subseteq \langle R\cdot T, S\cdot T\rangle \text{ holds by monotonicity }\ \} \\
& \langle y\cdot T\cdot x^\circ, \overline{yx}\cdot T\cdot x^\circ\rangle \subseteq y\overline{yx}\cdot T\cdot x^\circ \\
\equiv & \{\ \text{“split twist” rule (97) ; converses }\ \} \\
& \langle y\cdot T\cdot x^\circ, id\rangle \cdot (x\cdot T^\circ \cdot \overline{yx}^\circ) \subseteq \langle y, x\cdot T^\circ\rangle \cdot \overline{yx}^\circ \\
\equiv & \{\ x = x\cdot x^\circ\cdot x\ \} \\
& \langle y\cdot T\cdot x^\circ, id\rangle \cdot x\cdot x^\circ\cdot x\cdot T^\circ\cdot \overline{yx}^\circ \subseteq \langle y, x\cdot T^\circ\rangle \cdot \overline{yx}^\circ \\
\equiv & \{\ (84) \text{ below twice, since both } x\cdot x^\circ \text{ and } T^\circ \text{ are coreflexive }\ \} \\
& \langle y\cdot T\cdot x^\circ, x\cdot x^\circ\rangle \cdot x\cdot T^\circ\cdot \overline{yx}^\circ \subseteq \langle y, x\rangle \cdot T^\circ \cdot \overline{yx}^\circ \\
\equiv & \{\ (75) \text{ in which } f = x \text{ followed by (84) in which } \Phi = T\ \} \\
& (\langle y, x\rangle \cdot T\cdot x^\circ)\cdot (x\cdot T^\circ\cdot \overline{yx}^\circ) \subseteq \langle y, x\rangle \cdot T\cdot \overline{yx}^\circ \\
\equiv & \{\ (81)\ \} \\
& x \overset{T}{\rightharpoonup} y
\end{array}
$$

Two steps in the calculation above rely on fact

$$\langle R, S\rangle \cdot \Phi = \langle R, S\cdot \Phi\rangle \tag{84}$$

which is easy to justify:

$$
\begin{array}{ll}
& \langle R, S\rangle \cdot \Phi = \langle R, S\cdot \Phi\rangle \\
\equiv & \{\ \text{split pointfree definition }\ \} \\
& (\pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S)\cdot \Phi = \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S\cdot \Phi \\
\equiv & \{\ \text{converses and commutativity }\ \} \\
& \Phi\cdot (S^\circ\cdot \pi_2 \cap R^\circ\cdot \pi_1) = (\Phi\cdot S^\circ\cdot \pi_2)\cap (R^\circ\cdot \pi_1) \\
\Leftarrow & \{\ \text{fact } \langle \forall\ S, T\ ::\ R\cdot S\cap T = R\cdot (S\cap T)\rangle \equiv R\subseteq id \text{ in Ex.11.22 of [3] }\ \} \\
& \Phi \subseteq id
\end{array}
$$

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

$$y \leq x \Rightarrow x \overset{T}{\rightharpoonup} y \tag{85}$$

since

$$
\begin{array}{ll}
& x \overset{T}{\rightharpoonup} y \\
\Leftarrow & \{\ \text{conversion (82)}\ \}
\end{array}
$$

$$\begin{array}{ll} & x \stackrel{T}{\multimap} y \\ \Leftarrow & \quad \{ \text{ FD reflexivity (64) } \} \\ & y \leq x \end{array}$$

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

$$x \stackrel{R}{\multimap} y \Rightarrow x \stackrel{R}{\multimap} \overline{y} \tag{86}$$

which the reader may wish to prove as an exercise. All other MVD inference axioms can be found in [12], section 7.4.1. In this paper we don't go beyond this point.

12 Conclusions

This paper presents a pointfree version of functional dependency theory, the kernel of relational database design "à la Codd". Contrary to the intuition that a binary relation is just a particular case of n-ary relation, this paper shows the effectiveness of the former in "explaining" and reasoning about the latter.

It turns out that the theory becomes more general and considerably simpler. The adoption of the (pointfree) binary relation calculus is beneficial in several respects. First, the fact that pointfree notation abstracts from "points" or variables makes the reasoning more compact and effective. Elegant formulæ such as (41) — when compared with (3) — come in support of 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 [14] concerning a pointfree model of *hash tables*).

13 Future work

While addressing the foundations of FD theory in a pointfree style, no claim is made in this paper for extending or improving the standard theory. What is gained is a better starting point for relational database theory [12], a fairly large (and often convoluted) body of knowledge [6].

The effectiveness of the approach can only be tested once more and more results are dealt with. In this paper, multi-valued dependencies have only been hinted at. Join dependencies and difunctional dependencies [9] have not been considered at all. The use of functional dependencies in solving ambiguities in multiple parameter type classes in the Haskell type system [10] may happen to be another area of application of the reasoning techniques developed in this paper.

[6] FDs account for no more than 20% of pages in Maier's book [12].

At the level of the pointfree "transform" itself, our notion of *kernel* of a binary relation is a conservative one when compared to that of [7], which can be considered as an alternative (both coincide on functions). Moreover, *left* and *right conditions* [8] should be also exploited as alternatives to coreflexives in n-ary relation modelling. Finally, the connection between binary relation projection and Reynolds "relation on functions" expressed by (30) is worth studying in more detail, taking into consideration the corresponding point-free theory developed in [2].

Concerning representation theory and lossless decomposition, some recent results in [13] and [16] should be taken into account and generalized.

Acknowledgments

The work reported in this paper has been carried out in the context of the PURE Project *(Program Understanding and Re-engineering: Calculi and Applications)* funded by FCT (the Portuguese Science and Technology Foundation) under contract `POSI/ICHS/44304/2002`.

References

1. Chritiene Aarts, Roland Backhouse, Paul Hoogendijk, Ed Voermans, and Jaap van der Woude. A relational theory of datatypes, December 1992. Available from `www. cs. nott. ac. uk/~rcb/ papers`.
2. K. Backhouse and R.C. Backhouse. Safety of abstract interpretations for free, via logical relations and Galois connections. *Science of Computer Programming*, 15(1–2):153–196, 2004.
3. R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.
4. R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997. C.A. R. Hoare, series editor.
5. E.F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, June 1970.
6. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002. ISBN: 0-13-031995-3.
7. Jeremy Gibbons. When is a function a fold or an unfold?, 2003. Working document 833 FAV-12 available from the website of IFIP Working Group 2.1 57th meeting, New York City, USA.
8. Paul Hoogendijk. *A Generic Theory of Data Types*. PhD thesis, University of Eindhoven, The Netherlands, 1997.
9. A. Jaoua, S. Elloumi, A. Hasnah, J. Jaam, and I. Nafkha. Discovering Regularities in Databases Using Canonical Decomposition of Binary Relations. *JoRMiCS*, 1:217–234, 2004.
10. 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.

11. Akihiro Kanamori. The empty set, the singleton, and the ordered pair. *The Bulletin of Symbolic Logic*, 9(3):273–298, 2003.
12. D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983. ISBN 0-914894-42-0.
13. J.N. Oliveira. *Calculate databases with 'simplicity'*, September 2004. Presentation at the *IFIP WG 2.1 #59 Meeting*, Nottingham, UK.
14. J.N. Oliveira and C.J. Rodrigues. Transposing relations: from *Maybe* functions to hash tables. In *MPC'04 : 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.
15. 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.
16. 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.
17. Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.

A A little preorder construction

The concept of a *preorder* — ie. that of a *reflexive* and *transitive* endo-relation — is central to the mathematics of computing. It paves the way to Galois connections and other interesting topics (eg. lexicographic orders, etc.). In this annex we concentrate on a particular preorder construction which is used extensively in this paper. For more about preorders see eg. [1] and [4].

The construction. Let $A \xleftarrow{\sqsubseteq} A$ be a preorder. Given a function $A \xleftarrow{h} B$, the relation $B \xleftarrow{\preceq} B$ defined by

$$\preceq \stackrel{\text{def}}{=} h^\circ \cdot \sqsubseteq \cdot h \tag{87}$$

is also a preorder: it is reflexive,

$$\begin{array}{ll} & id \subseteq \preceq \\ \equiv & \quad \{ \text{ (87) and shunting (16) } \} \\ & h \subseteq \sqsubseteq \cdot h \\ \Leftarrow & \quad \{ \text{ } (\cdot h) \text{ is monotonic } \} \\ & id \subseteq \sqsubseteq \\ \equiv & \quad \{ \text{ } \sqsubseteq \text{ is reflexive } \} \\ & \text{TRUE} \end{array}$$

and transitive:

$$
\begin{array}{ll}
& \preceq \cdot \preceq \\
= & \quad \{ \text{ (87) twice; associativity of composition } \} \\
& h^\circ \cdot \sqsubseteq \cdot (h \cdot h^\circ) \cdot \sqsubseteq \cdot h \\
\subseteq & \quad \{ \text{ } h \text{ is simple (6) } \} \\
& h^\circ \cdot \sqsubseteq \cdot \sqsubseteq \cdot h \\
\subseteq & \quad \{ \text{ } \sqsubseteq \text{ is transitive } \} \\
& h^\circ \cdot \sqsubseteq \cdot h \\
= & \quad \{ \text{ (87) } \} \\
& \preceq
\end{array}
$$

Example. The *injectivity* preorder defined earlier on (37) is an example of this construction for $h = \mathsf{ker}$, $\preceq = \leq^\circ$ and $\sqsubseteq = \subseteq$:

$$\leq^\circ = \mathsf{ker}^\circ \cdot \subseteq \cdot \mathsf{ker} \tag{88}$$

that is,

$$\leq = \mathsf{ker}^\circ \cdot \subseteq^\circ \cdot \mathsf{ker}$$

(Note the extra converse operator.)

Preorder homomorphism. By construction, (87) establishes h as a preorder homomorphism — cf.

$$a \preceq a' \equiv h\ a \sqsubseteq h\ a'$$

in pointwise notation — which can be exploited to "lift" results from the $\sqsubseteq$ to the $\preceq$ order. We present two such results, one concerning monotonicity and the other Galois connections. For space economy, both will be presented restricted to endo-functions. (The general formulation is similar.)

Lifting monotone operators. Let $A \xleftarrow{k} A$ be a $\sqsubseteq$-monotonic endo-function, and k' be its h-counterpart, that is,

$$h \cdot k' = k \cdot h \tag{89}$$

Then k' is $\preceq$-monotonic:

$$
\begin{array}{ll}
& \preceq \subseteq k'^\circ \cdot \preceq \cdot k' \qquad (90) \\
\equiv & \quad \{ \text{ (87) twice ; (10) } \} \\
& h^\circ \cdot \sqsubseteq \cdot h \subseteq (h \cdot k')^\circ \cdot \sqsubseteq \cdot h \cdot k'
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{\ \text{(89) twice ; (10)}\ \} \\
& h^\circ \cdot \sqsubseteq \cdot h \subseteq h^\circ \cdot k^\circ \cdot \sqsubseteq \cdot k \cdot h \\
\Leftarrow & \{\ (h^\circ \cdot \_ \cdot h) \text{ is monotonic}\ \} \\
& \sqsubseteq \subseteq k^\circ \cdot \sqsubseteq \cdot k \\
\equiv & \{\ \text{since } k \text{ is } \sqsubseteq\text{-monotonic}\ \} \\
& \textsc{True}
\end{array}
$$

Examples. From

$$\mathsf{ker}\,(R \cdot T) = T^\circ \cdot (\mathsf{ker}\,R) \cdot T \tag{91}$$

we identify, for $h = \mathsf{ker}$, $k' = (\ \cdot T)$ and $k = (T^\circ \cdot \_ \cdot T)$. Since k is $\subseteq$-monotonic, from (90) we draw that k' is $\leq^\circ$-monotonic, which is equivalent to being $\leq$-monotonic. This justifies equation (40) in the main body of the paper. A similar argument can be provided to justify $\leq$-monotonicity of any relator F,

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

for $k = k' = \mathsf{F}$, since F is $\subseteq$-monotonic and

$$\mathsf{ker}\,(\mathsf{F}\,R) = \mathsf{F}(\mathsf{ker}\,R)$$

holds.

Lifting Galois connections. Suppose that functions $A \xleftarrow{k,j} A$ are Galois connected via preorder $\sqsubseteq$ and that k', j' are the h-counterparts of lower-adjoint k and upper-adjoint j, respectively. That is, facts

$$k^\circ \cdot \sqsubseteq = \sqsubseteq \cdot j \tag{92}$$

and

$$h \cdot k' = k \cdot h \tag{93}$$

$$h \cdot j' = j \cdot h \tag{94}$$

hold. Then k', j' are $\preceq$-Galois connected,

$$k'^\circ \cdot \preceq = \preceq \cdot j' \tag{95}$$

as proved below:

$$
\begin{array}{ll}
& k'^\circ \cdot \preceq \\
= & \{\ (87)\ \} \\
& k'^\circ \cdot h^\circ \cdot \sqsubseteq \cdot h
\end{array}
$$

$$
\begin{array}{ll}
= & \{\ \text{(93) and converses}\ \} \\
& h^\circ \cdot k^\circ \cdot \sqsubseteq \cdot h \\
= & \{\ \text{(92) followed by (94)}\ \} \\
& h^\circ \cdot \sqsubseteq \cdot h \cdot j' \\
= & \{\ (87)\ \} \\
& \preceq \cdot j'
\end{array}
$$

Examples. Consider the following instances of T in (91) and corresponding instances of k, k', j, j', for some function g of appropriate type:

$$
\begin{array}{l}
T := g \quad \begin{cases} j' = (\ \cdot g) \\ j = (g^\circ \cdot \_ \cdot g) \end{cases} \\
T := g^\circ \quad \begin{cases} k' = (\ \cdot g^\circ) \\ k = (g \cdot \_ \cdot g^\circ) \end{cases}
\end{array}
$$

The fact that k and j are Galois connected stems from the composition of shunting rules (16,17):

$$g \cdot X \cdot g^\circ \subseteq Y \equiv X \subseteq g^\circ \cdot Y \cdot g$$

Then, from (95) we draw

$$k'^\circ \cdot \leq^\circ = \leq^\circ \cdot j'$$

which, taking converses, is the same as

$$j'^\circ \cdot \leq = \leq \cdot k'$$

that is,

$$(\ \cdot g) \cdot \leq = \leq \cdot (\ \cdot g^\circ)$$

— ie. (55) — holds.

A similar argument will justify Galois connection (47), stemming from relational *split* being $\ker$-homomorphic to relational *meet* (45), which is the upper-adjoint in its defining Galois connection:

$$T \subseteq R \cap S \equiv T \subseteq R \wedge T \subseteq S \tag{96}$$

Because of the extra converse in $\leq^\circ$ in (88), the fact that *meet* is the upper-adjoint wrt. $\subseteq$ casts *split* as the lower-adjoint wrt. $\leq$:

$$\langle R, S \rangle \leq T \equiv R \leq T\ \wedge\ S \leq T$$

A more explicit argument is as follows:

$$\begin{array}{ll} & \langle R, S\rangle \leq T \\ \equiv & \quad \{ \text{ (37) and (45) } \} \\ & \ker T \subseteq (\ker R) \cap (\ker S) \\ \equiv & \quad \{ \text{ (96) } \} \\ & \ker T \subseteq \ker R \ \wedge \ \ker T \subseteq \ker S \\ \equiv & \quad \{ \text{ (37) twice } \} \\ & R \leq T \ \wedge \ S \leq T \end{array}$$

B The "split twist" rule

A step in the proof of lossless decomposition (83) is based on the following equivalence,

$$\langle R, S\rangle \subseteq \langle U, V\rangle \cdot X \equiv \langle R, id\rangle \cdot S^\circ \subseteq \langle U, X^\circ\rangle \cdot V^\circ \tag{97}$$

itself a consequence of

$$\langle R, S\rangle \cdot T \subseteq \langle U, V\rangle \cdot X \equiv \langle R, T^\circ\rangle \cdot S^\circ \subseteq \langle U, X^\circ\rangle \cdot V^\circ \tag{98}$$

for $T := id$. In order to prove (98), we reason using points x, y and z:

$$\begin{array}{ll} & (y, z)\ \langle R, S\rangle \cdot T\ x \\ \equiv & \quad \{ \text{ composition and split } \} \\ & \langle \exists\, u \ ::\ y\ R\ u \ \wedge \ z\ S\ u \ \wedge \ u\ T\ x\rangle \\ \equiv & \quad \{ \text{ converses } \} \\ & \langle \exists\, u \ ::\ y\ R\ u \ \wedge \ x\ T^\circ\ u \ \wedge \ u\ S^\circ\ z\rangle \\ \equiv & \quad \{ \text{ split and composition } \} \\ & (y, x)\ \langle R, T^\circ\rangle \cdot S^\circ\ z \end{array}$$

Similarly,

$$(y, z)\ \langle U, V\rangle \cdot X\ x \equiv (y, x)\ \langle U, X^\circ\rangle \cdot V^\circ\ z$$

and so on.