JOSÉ NUNO FONSECA DE OLIVEIRA
Departamento de Informática
Universidade do Minho

Computing theories refactoring via the PF-transform: the data dependency case study

Sumário da lição de síntese preparada para provas de Agregação nos termos do Decreto-Lei n.º 239/2007 de 19 de Junho.

Braga, Dezembro de 2008

Computing theories refactoring via the PF-transform: the data dependency case study

José Nuno Fonseca de Oliveira

Dezembro de 2008

Contents

About this Document

Portuguese Act 239/07 dated June 19th, 2007, establishes conditions for academics to obtain the "Agregação" title, a degree in the Portuguese academic system comparable to the Habilitation in other European countries.

According to clause (c) of article 5 and number 4 of article 13 of the same act, a seminar or lecture must be delivered by the candidate, addressing a topic in the scope of the chosen field of knowledge or specialization area. This talk is part of an examination process which should assess the merit of the scientific work of the candidate, his/her research skills and his/her ability to carry out independent research.

This document is provides the summary of the intended lecture required by clause (c) of article 8 of Act 239/07. The topic chosen for the lecture has to do with recent research concerns and teaching efforts of the candidate. In broad terms, the talk will address an area of fundamental interest to the success of software engineering as a scientific body of knowledge: the ability to *calculate* programs from abstract models. More specifically, it focuses on a particular theory of great relevance in computing since it started three decades ago: data dependency theory used in relational database design.

This two-tiered structure of the talk is intended for an audience of both specialists and non-specialists. Readers are referred to an accompanying paper [35] recently submitted to an international journal in the field, for many technical details which are omitted for economy of presentation.

For a detailed explanation of the antecedents of the approach and pedagogical considerations on how to incorporate it in computing curricula, based on the local experience at Minho University in the last twenty years, readers are also referred to "twin" report [32] submitted for the fulfillment of clause (b) of article 5 of Act 239/07.

1 Introduction

> *"Certaines personnes ont [l'affectation] d'éviter en apparence toute espèce de calcul, en traduisant par des phrases fort longues ce qui s'exprime très brièvement par l'algèbre, et ajoutant ainsi à la longueur des opérations, les longueurs d'un langage qui n'est pas fait pour les exprimer. Ces personnes-là sont en arrière de cent ans."*
>
> Evariste Galois (1831)

This lecture has to do with the foundations of software engineering. Why bother about such foundations? To begin with, let us inquire ourselves about the phrase *software engineering* itself. The terminology seems to date from the Garmisch NATO conference in 1968, from whose report [26] we quote the following excerpt:

> *In late 1967 the Study Group recommended the holding of a working conference on Software Engineering. The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.*

Provocative or not, the need for sound theoretical foundations has clearly been under concern since the very beginning of the discipline. However, how "scientific" do such foundations turn out to be, now that four decades have since elapsed?

In an excellent essay on the origins of scientific technology, Russo [41] establishes a simple criterion to verify whether a particular technology is "scientific" or not: just check whether one can compile a manual of exercises for it; if this is not possible, it's certainly not a scientific body of knowledge. Such a seemingly naïve principle is made more precise in the following quotation excerpted from [41] and illustrated in Fig. 1:

> *The immense usefulness of exact science consists in providing models of the real world within which there is a guaranteed method for telling false statements from true. (...) Such models, of course, allow one to describe and predict natural phenomena, by translating them to the theoretical level via correspondence rules, then solving the "exercises" thus obtained and translating the solutions obtained back to the real world. (...) There is, however, another possibility, much more interesting: moving freely within the theory, and so reaching points not associated to anything concrete by correspondence rules. From such a point in the theoretical model one can often construct the corresponding reality, thus modifying the existing world.*

It is the second part of this excerpt — which tallies with the witty remark *"scientists discover the world that exists; engineers create the world that never was"* by aerospace engineer Theodore Von Karman [1] — that leads us from science to technology. Put in other words, science is about understanding how (existing) things work and technology is about ensuring that some desirable (new, or previously unknown) things happen reliably. Properties of real-world entities are identified which, once expressed by mathematical formulæ, become abstract models which can be queried and reasoned about.

The recent terminological explosion in the software engineering field rooted on the word "model" [2] clearly tells software designers aware of the relevance of (abstract) modeling. However, is such an emphasis on modeling enough to entrust software design as a scientific technology? Certainly not, the main problem residing in the fact that it is hard (if not at all impossible) to reason about (="solve exercises" in Russo's terms) many such models.

This is surely a handicap of widespread modeling techniques based on (naïve) pictures, such as happens with entity-relationship (ER) diagrams [12] in database design and with several kinds of diagram in the UML [11], for instance. By contrast, Petri nets [40] provide an example of modeling technique based on pictures (graphs) which do have a theoretical meaning amenable to formal reasoning. However, even in the golden world of mathematically sound models life is not at all easy. The modeling strategy itself raises a kind of *notation* conflict between *descriptiveness* (ie., adequacy to describe domain-specific objects and

[1] This remark is quoted from the version of `http://www.discoverengineering.org` available at the time of writing.

[2] See for instance UML (="Unified Modeling Language"), MDE (="model driven engineering"), MDA (="model driven architecture"), MBT (="model-based testing") and so on.

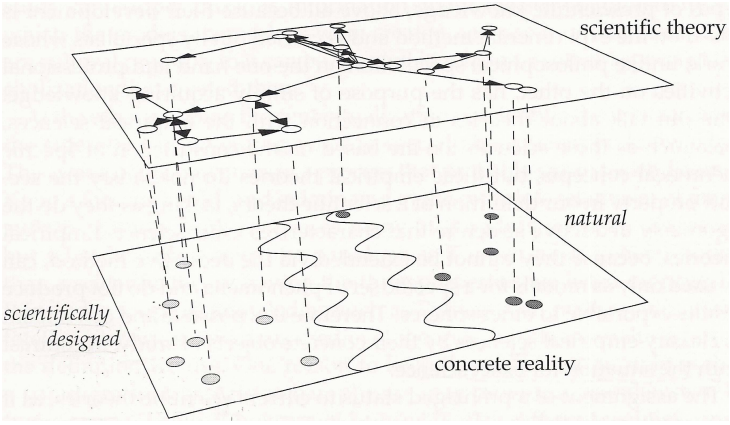


Figure 1: Picture extracted from [41] where its caption goes as follows: *The role of scientific technology. Dark-shaded circles on the concrete (lower) plane represent objects from nature or prescientific technology. Their counterparts on the theoretical (upper) plane are linked via logical deductions (arrows) to many other constructs, which may or may not have a concrete counterpart. Some of these theoretical constructs give rise, via correspondence rules (dashed lines), to new concrete objects (lightly shaded circles on the lower plane).*

properties, inc. diagrams or other graphical objects) and *compactness* (as required by algebraic reasoning and calculation "exercises").

Take the fields of programming language semantics and program refinement, for instance. Both are expressed in logics rooted on the predicate calculus and naïve set theory, leading to models of both specifications and programs and to proof rules helping to move from the former to the latter. However, just glance through a textbook or paper on such subjects and compare the visual aspect of maths displays with those of a book on physics or engineering mathematics: the former will look far more complex than the latter; "exercises" won't be solved by simple and elegant calculations. Altogether, the theories behind software don't look smart enough.

What can we do, then, faced with such (apparently) immature theories? What's our advice to the eager software practitioner trying to discharge, for instance, a complex proof which emerges from the application of one such theory to a real-life situation? There are essentially two ways to proceed. One is typical of the computer age in which we live today: just ask a theorem prover or model checker to help you; the other (more akin to traditional paper and pencil reasoning) will try and reduce the complexity of the argument to be proven, so that each step in such a (reduced) proof becomes "size-minded".

Clearly, such a reduction effort has to go further than merely re-structuring the proof in lemmas and auxiliary results. The main problem resides in the fact that computer programs, and the formal logics used to reason about them, involve too "fine-grained" notations which quantify over "too little". For instance, suppose one's problem is to find a program able to perform some particular task, eg. sorting. In the scientific method, such a program should *emerge* as solution to some equation prescribing its behaviour (model). Such an equation would quantify over programs and specs, of course. However, when it comes to doing the actual work, one is dumped to a lower level of abstraction where variables and quantifiers range over data values such as linked lists, pointers, array locations and the like. It is at this lower *pointwise* level that the models of facts to prove grow too detailed, enormous and unmanageable.

1.1 Need for transforms

The kind of notational problem mentioned just above is not wholly new in the history of science. Elsewhere in physics and several branches of engineering, for instance, people have learned to overcome lack of calculation agility by changing "mathematical space", for instance by moving (temporarily) from the t-space (t for time) to the s-space in the *Laplace transformation* (fig. 2). Quoting Kreyszig [22], p.242:

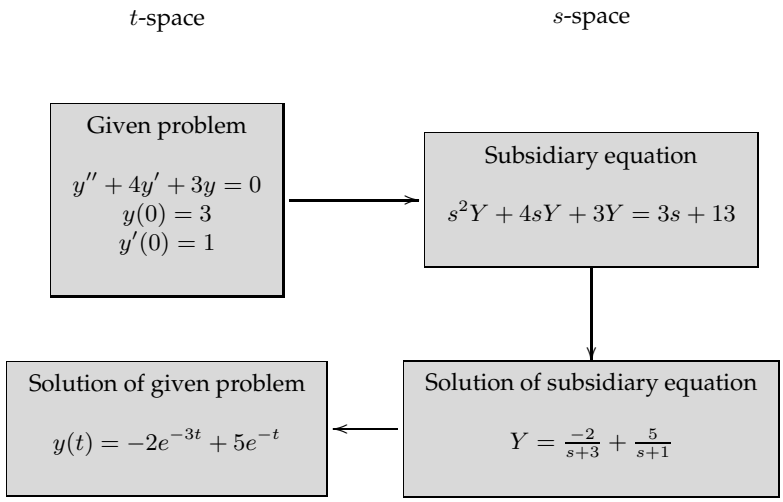


Figure 2: Example of Laplace transformation (quoted from [22]).

The Laplace transformation is a method for solving differential equations (...) The process of solution consists of three main steps:

1st step. *The given "hard" problem is transformed into a "simple" equation (*subsidiary equation*).*

2nd step. *The subsidiary equation is solved by* ***purely algebraic*** *manipulations.*

3rd step. *The solution of the subsidiary equation is transformed back to obtain the solution of the given problem.*

In this way the Laplace transformation reduces the problem of solving a differential equation to an ***algebraic problem****.*

Note Kreyszig's notion of *complexity reduction*: the original problem model reduces to an *algebraic* model which is solved by ordinary school algebra. The question arises: is there a s-space equivalent for the predicate calculus?

That the integral/differential calculus and the predicate calculus may have something in common can be observed by putting, for instance, the following two expressions side by side [3]:

$$\langle \int x \ : \ 0 \leq x \leq 10 : \ x^2 - x \rangle$$
$$\langle \forall \, x \ : \ 0 \leq x \leq 10 : \ x^2 \geq x \rangle$$

However, we cannot infer from this notation analogy that a formal correspondence exists between the two calculi. What we can do is to try and find an algebraic space such that *(a)* predicates can be translated into algebraic expressions; *(b)* quantifiers disappear along the translation process by imploding into operations of the target algebraic calculus; *(c)* the laws in such a calculus share the spirit and shape of school algebra captured by easy-to-use rules such as eg.

$$x + y \leq z \quad \Leftrightarrow \quad x \leq z - y \tag{1}$$

[3] We adopt the Eindhoven quantifier notation and calculus [5, 4] whereby $\langle \forall \, x \ : \ R : \ T \rangle$ and $\langle \exists \, x \ : \ R : \ T \rangle$ mean, respectively *"for all x in range R it is the case that T"* and *"there exists at least one x in range R such that T"*.

which enables one to "shift" symbols (eg. y just above) or sub-expressions from one side of an inequality to the other by changing signs.

Why do we adhere to this kind of rule? Surely there is a cultural factor: such rules have been used for centuries in calculations, since (at least) the "al-gabr" rule of *On the calculus of al-gabr and al-muqâbala* by Abû Al-Huwârizmî, the famous 9c Persian mathematician from whose name words such as *algebra*, *algorithm* etc. have been coined.

It was with great excitement that European mathematicians re-discovered this calculation style in the 16c (see eg. [28]) as a kind of *transform* of classical geometry-explicit reasoning. Much later, "al-gabr" rules were found to generalize to other areas of mathematical reasoning and became known as Galois connections [38], after the work of the famous French mathematician. It is only at this stage that their potential for calculation and genericity are eventually appreciated. For instance, instead of providing an explicit definition for integer division (eg. as a while-loop or recursive function), one may calculate with its defining property

$$\langle \forall\, d, n, q \in \mathbb{N} \;::\; q \times d \leq n \;\Leftrightarrow\; q \leq n/d \rangle \qquad \begin{array}{c|c} n & d \\ \hline r & q \end{array} \tag{2}$$

that is, with the "al-gabr" rule (Galois connection) which defines its behaviour. (See [42] for the derivation of the integer division algorithm based on (2) alone.)

Why can one be so confident of the accuracy of such implicit definition? From school we know that n/d is the largest whole number q (quotient) such that $q \times d$ approximates n, the difference being referred to as the *remainder*. Note that this is *precisely* what (2) means: by reading the equivalence from right to left ($\Leftarrow$) and substituting $q := n/d$ we obtain $q \times (n/d) \leq n$ meaning that n/d is one such approximation; by reading it from left to right, we obtain implication $q \times d \leq n \;\Rightarrow\; q \leq n/d$, which means that n/d is largest among all such approximations q.

2 Introducing the PF-transform

Rules such as (2) are not exclusive to number theory. Moving away from numbers, let us consider facts such as

`"a"`	$isPrefixOf$	`"ab"`
Archimedes	$diedIn$	Syracuse
TRUE	$\in$	$\{$TRUE, FALSE$\}$

all captured by the idea of a *binary relation* (between strings, between of people and towns and between Booleans and sets of Booleans, in the examples given).

No other concept traverses human knowledge more ubiquitously than that of a relation, from philosophy to mathematics, to information systems — think eg. of relational databases — and so and so on. Let us, in general, write $b\ R\ a$ to denote the fact that item b is related to item a in relation R, that is, that pair (b, a) is in R. Let $R \cdot S$ denote the composition of R and S (read "$R \cdot S$" as "R after S") defined in the obvious way: $b(R \cdot S)c$ holds wherever there exists at least one mediating a such that bRa and aSc both hold.

It is easy to check that $R \cdot S$ has a *multiplicative* flavour: it is associative (albeit not commutative), it distributes over the union of two relations ($R \cup S$) and it has a unit element, the identity relation id defined in the obvious way: $b\ id\ a$ iff $b = a$. Given such a multiplicative flavour, one may question: is there any reasonable notion of *relation division* which one could put in parallel with (2)? It turns out that one just has to re-interpret $\leq$ in (2) as relation inclusion and write:

$$\langle \forall\, X, S, R \;::\; R \cdot X \subseteq S \;\Leftrightarrow\; R \subseteq S/X \rangle \tag{3}$$

where X, S and R refer to binary relations. Now, what does S/X mean? By replaying our reading of (2) above, it will be the largest relation whose pre-composition with X (best) approximates S. Can one write

this in a more tangible way? It can be shown that S/X is the relation whose pointwise meaning is

$$a(S/X)b \quad \Leftrightarrow \quad \langle \forall\, c \,:\, b\, X\, c :\, a\, S\, c \rangle \tag{4}$$

The economy of notation S/X compared to its expansion as a universal quantification is obvious, as happens with the expansion of relational composition into the existential quantification implicit in its definition,

$$b(R \cdot S)c \quad \Leftrightarrow \quad \langle \exists\, a \,::\, bRa \,\wedge\, aSc \rangle \tag{5}$$

and with that of relation inclusion as another universal quantification:

$$R \subseteq S \quad \Leftrightarrow \quad \langle \forall\, a, b \,:\, b\, R\, a :\, b\, S\, a \rangle \tag{6}$$

Note how the right hand sides of the three maths displays above already fulfill one of our wishes recorded earlier on about a transform for the predicate calculus similar to the Laplace transform: the quantified expressions disappear once variables are dropped, imploding into operators of the target calculus. Moreover, reasoning about such combinators dispenses with their quantification counterparts: 'pointfree" rules such as eg. (3) are enough [4].

The phrase *Pointfree transform* (or *PF-transform* for short) will hereafter denote this process of transforming predicate calculus expressions into their equivalent binary relation representations. Given a binary predicate $p(b, a)$ we will denote by $[\![p]\!]$ the binary relation such that $b[\![p]\!]a \;\Leftrightarrow\; p(b, a)$ holds, for all suitably typed a and b. We have a problem, though: how do we transform a unary predicate u into a *binary* relation? We just build the relation $[\![u]\!]$ such that

$$b[\![u]\!]a \quad \Leftrightarrow \quad (b = a) \,\wedge\, (u\, a) \tag{7}$$

holds. That is, $[\![u]\!]$ is the relation that maps every a which satisfies u (and only such a) onto itself. Clearly, such relation is a fragment of the identity relation: $[\![u]\!] \subseteq id$.

Relations at most id are referred to as *coreflexive* relations [5]. They are extremely useful in calculations because they act as data filters. For instance, suppose we need to transform the following variant of (5)

$$\langle \exists\, a \,:\, u\, a :\, b\, R\, a \,\wedge\, a\, Sc \rangle$$

where predicate u establishes the range of the quantification. It can be easily checked that

$$R \cdot [\![u]\!] \cdot S \tag{8}$$

is the desired extension to relational composition. Coreflexives can also model sets in the obvious way: the PF-meaning of a set S is coreflexive $[\![\lambda a.a \in S]\!]$, that is,

$$b[\![S]\!]a \quad \Leftrightarrow \quad (b = a) \,\wedge\, a \in S \tag{9}$$

[4] Already in the case of integer division it can be shown that (2) alone is enough to calculate properties such as eg. $(n/m)/d \;=\; n/(d \times m)$, for instance, dispensing with suprema reasoning, remainders and the like [43].

[5] Some standard terminology arises from the id relation: a (endo) relation R (often called an *order*) will be referred to as *reflexive* iff $id \subseteq R$ holds and as *coreflexive* iff $R \subseteq id$ holds.

The following table [6]

Pointwise	Pointfree
$\langle \exists\ a\ ::\ b\ R\ a\ \wedge\ a\ S\ c \rangle$	$b(R \cdot S)c$
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \forall\ x\ ::\ x\ R\ b \Rightarrow x\ S\ a \rangle$	$b(R \setminus S)a$
$\langle \forall\ c\ ::\ b\ R\ c \Rightarrow a\ S\ c \rangle$	$a(S\ /\ R)b$
$b\ R\ a\ \wedge\ c\ S\ a$	$(b, c)\langle R, S \rangle a$
$b\ R\ a\ \wedge\ d\ S\ c$	$(b, d)(R \times S)(a, c)$
$b\ R\ a\ \wedge\ b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ b)\ R\ (g\ a)$	$b(f^\circ \cdot R \cdot g)a$
TRUE	$b \top a$
FALSE	$b \bot a$

(10)

includes the most common relational operators associated to the PF-transform. Lowercase symbols (eg. f, g) stand for relations which are *functions* and R° denotes the *converse* of R, that is, the relation such that $a(R^\circ)b$ holds iff bRa holds. (By the way, note that converse is involutive

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

and commutes with composition

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{12}$$

in a contravariant way.) The two variants of division "/" and "\" in (10) arise from the fact that relation composition is not commutative, the "al-gabr" rule for $R \setminus S$ being similar to (3):

$$R \cdot X \subseteq S \quad \Leftrightarrow \quad X \subseteq R \setminus S \tag{13}$$

Divisions involving functions can be expressed via composition and converse alone, since $h \setminus R = h^\circ \cdot R$ and $R\ /\ h^\circ = R \cdot h$ hold. So, the "al-gabr" rules for functions are easier to express [7]

$$f \cdot R \subseteq S \quad \Leftrightarrow \quad R \subseteq f^\circ \cdot S \tag{14}$$

$$R \cdot f^\circ \subseteq S \quad \Leftrightarrow \quad R \subseteq S \cdot f \tag{15}$$

and bear particular resemblance with school algebra: like numbers in (1), functions can be shifted around in relational expressions by "changing sign" (which in the relational context means taking converses).

Relations $\top$ ("top") and $\bot$ ("bottom") are respectively the largest (smallest) relations of their type. But, what do we mean by the *type* of a relation? The answer can be found in the section which follows.

3 Related Work

The idea of encoding predicates in terms of relations was initiated by De Morgan in the 1860s and followed by Peirce who, in the 1870s, found interesting equational laws of the calculus of binary relations [39]. The pointfree nature of the notation which emerged from this embryonic work was later further exploited by Tarski and his students [44]. In the 1980's, Freyd and Ščedrov [16] developed the notion of an *allegory* (a category whose morphisms are partially ordered) which finally accommodates the binary relation calculus

[6] We will drop brackets $[\![\ ]\!]$ wherever clear from the context.

[7] These are often referred to as *shunting rules* [10].

as special case. In this context, a relation R is viewed as an arrow (morphism) $B \xleftarrow{R} A$ between sets B and A, respectively referred to as the target and source types of R. Composition of such arrows corresponds to relational composition (5), identity is id, and relational expressions can be "type-checked" by drawing diagrams such as in category theory.

Such advances in mathematics were meanwhile captured by the Eindhoven computer science school in their development of program construction as a mathematical discipline [1, 6, 14, 10, 5] enhanced by judicious use of Galois connections, as already illustrated above.

Our view of this approach as a kind of Laplace transform for logics was first expressed in [29]. Such a transform (the PF-transform) has henceforth been applied to several areas of the software sciences, namely relational database schema design [30, 2, 13], *hashing* [36], software components [7], coalgebraic reasoning [8], algorithmic refinement [37], data refinement [13, 34], separation logic [46] and extended static checking [33].

The remainder of the lecture will be devoted to yet another example of application of the PF-transform which we have been studying since [31] and which we regard as a particularly expressive illustration of the power of the PF-transform: data dependency theory [25]. This theory, which is at the heart of relational database design, is *pointwise* (as most theories in computing are). In [35] we explain how to "re-factor" such a theory via the PF-transform, leading to a calculational style instead of reasoning about (sets of) tuples in conventional "implication-first" logic style.

It turns out that the theory becomes more general, more structured and 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. Pointfree re-factoring also leads to a generalization of data dependency theory which paves the way to interesting synergies with other branches of computer science. In the sequel we provide a glimpse of [35] which omits many technical details in order to retain what really matters: the evidence that computing theories are indeed sharpened by PF-transformation.

4 Applying the PF-transform to data dependency

In database design, the complex structure of the objects and entities to be modelled demands much on descriptiveness, thus entailing the need for graphical notations (already mentioned) and verbose programming notations such as Cobol [3] and SQL [20]. When it comes to reasoning about the semantics of such diagrams or notations, predicate/temporal logics and naïve set theory are the most common formal resources. However, such *pointwise* notations involving operators as well as variable symbols, logical connectives, quantifiers, etc. are not handy enough for calculations.

In standard relational data processing, real life objects or entities are recorded by assigning values to their observable properties or *attributes*. A database file is a collection of such attribute assignments, one per object, such that 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 just a collection of several such n-ary relations.

Data dependency theory is based essentially on two concepts: that of a functional dependency (FD) and that of a multi-valued dependency (MVD). Both are central to standard database theory, where they addressed in an axiomatic way. Maier [25] provides the following definition for FD-satisfiability:

Definition 1 *Given subsets $x, y \subseteq S$ of the relation scheme S of a n-ary 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,*

$$\langle \forall\, t, t' \;:\; t, t' \in R : \quad t[x] = t'[x] \quad \Rightarrow \quad t[y] = t'[y] \;\rangle \tag{16}$$

(Notation $t[a]$ adopted in (16) means "the value exhibited by attribute a in tuple t".)
□

Formula (16), with its logical implication inside a two-dimensional universal quantification, is not particularly graceful. Designs involving many FDs at the same time would be hard to reason about if based on (16) alone. This situation gets worse when the more general (and useful) concept of a *multi-valued* dependency (MVD) is addressed. This is defined by Maier [25] as follows:

Definition 2 *Given subsets $x, y \subseteq S$ of the relation scheme S of an n-ary 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 x and y and "agrees" with t' on $z = S - xy$, that is,*

$$\langle \forall\, t, t' \,:\, t, t' \in R : \quad t[x] = t'[x] \quad \rangle \qquad (17)$$
$$\Downarrow$$
$$\langle \exists\, t'' \,:\, t'' \in R : \quad t[xy] = t''[xy] \wedge \quad \rangle$$
$$t''[z] = t'[z]$$

holds. □

Reference [9] gives the alternative definition of MVD which follows:

Definition 3 *Given subsets $x, y \subseteq S$ of the relation scheme S of an n-ary relation R, let $z = S - xy$. R is said to satisfy the* multi-valued *dependency (MVD) $x \twoheadrightarrow y$ iff, for every xz-value ab, that appears in R, one has $Y(ab) = Y(a)$, where for every $k \subseteq S$ and k-value c, function Y is defined as follows:*

$$Y(c) \quad = \quad \{v \mid \langle \exists\, t \,:\, t \in R :\, t[k] = c \,\wedge\, t[y] = v \rangle\}$$

□

Notation is overly simplified in this definition. In fact, function Y should be equipped with two extra parameters, attribute k and relation R itself. So, the precise assertion that R satisfies MVD $x \twoheadrightarrow y$ is

$$\langle \forall\, a, b \,:\, \langle \exists\, t \,:\, t \in R :\, t[xz] = ab \rangle :\, Y_{R,x}(a) = Y_{R,xz}(ab) \rangle \qquad (18)$$

as is illustrated in the following picture:

	x	y	z
t	a	c	b
t''	a	c	b'
t'	a	c'	b'
t'''	a	c'	b

(19)

Despite its complexity, the MVD concept is central to one of the main ingredients of relational data refinement: the principle of *lossless decomposition* [25] whereby complex data models can be factored into relational databases. Such a complexity has lead database theorists to develop FD/MVD-theory in an axiomatic style, based on the so-called *Armstrong axioms*, which can be used as inference rules for such dependencies. Equivalent axioms have been found which make FD/MVD checking more efficient. However, most database practitioners use this theory while ignoring its foundations. Even textbooks such as [45] and [17] do not go very deep into the subject. Can this be accepted?

Our approach is to regard such standard set-theory-formulated database concepts as "hard" problems (in the sense of [22]) to be transformed into "simple", *subsidiary equations* dispensing with points and involving only binary relation combinators. As in the Laplace transformation, these are solved by *purely algebraic* manipulations and the outcome is mapped back to the original (descriptive) mathematical space wherever required.

Note the advantages of this two-tiered approach: intuitive, domain-specific descriptive formulæ are used wherever the model is to be "felt" by people. Such formulæ are transformed into a more *elegant*, simple

and compact — but also more cryptic — algebraic notation whose single purpose is easy manipulation in calculations.

First of all, we need to settle some notation. Let R be a n-ary relation with schema S, t be a tuple in R and a be an attribute in S. Tuples can be regarded as inhabitants of n-dimensional Cartesian products, either in the standard format (eg. $A_1 \times \cdots \times A_n$) or in "rich syntax format" equipped with tuple constructors and selector (field) names, one per attribute. From our perspective, it doesn't matter which of these alternatives is adopted, since in *both cases* attributes are modelled by *(projection) functions*. Since this view extends smoothly to collections of attributes, we can regard x and y in (16) as functions and write:

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

Assuming the universal quantification implicit, we launch PF-transformation as follows:

$$\begin{array}{ll}
 & t \in R \ \wedge\ t' \in R \ \wedge\ (x\ t) = (x\ t') \quad \Rightarrow \quad (y\ t) = (y\ t') \\
\Leftrightarrow & \qquad \{\ \text{PF-transform rule}\ \ b(f^\circ \cdot R \cdot g)a \Leftrightarrow (f\ b)R(g\ a) \quad \text{twice}\ \ (10)\ \} \\
 & t \in R \ \wedge\ t' \in R \ \wedge\ t(x^\circ \cdot x\ )t' \quad \Rightarrow \quad t(y^\circ \cdot y\ )t' \\
\Leftrightarrow & \qquad \{\ (9)\ \text{twice}\ \} \\
 & t = u \ \wedge\ t[\![R]\!]u \ \wedge\ t' = u' \ \wedge\ t'[\![R]\!]u' \ \wedge\ t(x^\circ \cdot x\ )t' \quad \Rightarrow \quad t(y^\circ \cdot y\ )t' \\
\Leftrightarrow & \qquad \{\ \wedge\ \text{is commutative; substitution of equals for equals; coreflexives}\ \} \\
 & t[\![R]\!]u \ \wedge\ u(x^\circ \cdot x\ )u' \ \wedge\ u'[\![R]\!]^\circ t' \quad \Rightarrow \quad t(y^\circ \cdot y\ )t' \\
\Leftrightarrow & \qquad \{\ \text{going pointfree via composition and relation inclusion (6)}\ \} \\
 & [\![R]\!] \cdot (x^\circ \cdot x\ ) \cdot [\![R]\!]^\circ \quad \subseteq \quad y^\circ \cdot y \\
\Leftrightarrow & \qquad \{\ \text{rules (14) and (15)}\ \} \\
 & y\ \cdot [\![R]\!] \cdot x^\circ \cdot x\ \cdot [\![R]\!]^\circ \cdot y^\circ \quad \subseteq \quad id \\
\Leftrightarrow & \qquad \{\ \text{converse versus composition (12) followed by (20) below}\ \} \\
 & \mathit{img}\,(y\ \cdot [\![R]\!] \cdot x^\circ) \quad \subseteq \quad id
\end{array}$$

The step just above introduces the *image* operator on relations

$$\mathit{img}\, R \quad = \quad R \cdot R^\circ \tag{20}$$

which is useful in characterizing two properties of relations: *surjectivity*, which holds on a relation R wherever $\mathit{img}\, R$ is reflexive, and *simplicity*, which holds wherever $\mathit{img}\, R$ is coreflexive.

Based on this terminology, we can restate definition 1 as follows: a *n-ary* relation R as in definition 1 satisfies FD $x \rightarrow y$ iff the *y, x-projection* of $[\![R]\!]$

$$y\ \cdot [\![R]\!] \cdot x^\circ \tag{21}$$

is simple.

There is another, even simpler alternative to this pointfree FD definition, which diverts from the calculation above in the last two steps:

$$\begin{array}{ll}
 & [\![R]\!] \cdot (x^\circ \cdot x\ ) \cdot [\![R]\!]^\circ \quad \subseteq \quad y^\circ \cdot y \\
\Leftrightarrow & \qquad \{\ \text{composition is associative}\ \} \\
 & ([\![R]\!] \cdot x^\circ) \cdot (x\ \cdot [\![R]\!]^\circ) \quad \subseteq \quad y^\circ \cdot y
\end{array}$$

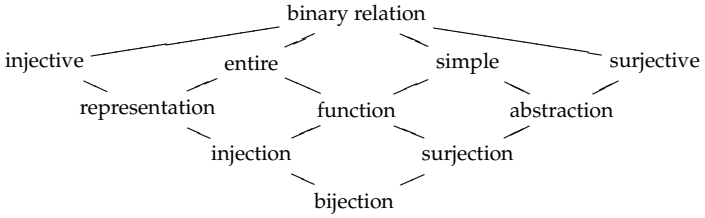


Figure 3: Binary relation taxonomy

$$
\begin{array}{ll}
\Leftrightarrow & \{ \text{ converses (11, 12) } \} \\
& (x \cdot [\![R]\!]^\circ)^\circ \cdot (x \cdot [\![R]\!]^\circ) \quad \subseteq \quad y^\circ \cdot y \\
\Leftrightarrow & \{ \text{ introduce ordering } \leq \text{ defined below (22) } \} \\
& y \quad \leq \quad x \cdot [\![R]\!]^\circ \\
\Leftrightarrow & \{ \text{ converse of coreflexive is itself (symmetry) } \} \\
& y \quad \leq \quad x \cdot [\![R]\!]
\end{array}
$$

The pre-ordering on relations resorted to above,

$$R \leq S \quad \Leftrightarrow \quad ker\, S \subseteq ker\, R \tag{22}$$

compares the *kernels* of both relations in reverse order, where

$$ker\, R \quad = \quad R^\circ \cdot R \tag{23}$$

provides a measure of how defined and/or injective a relation is. In particular, a relation is said to be *entire* (a term preferred to *totally defined* in the standard terminology [10]) iff its kernel is reflexive and *injective* iff its kernel is coreflexive.

Clearly, $R \leq S$ means that R is *less injective or more defined* than S, since *ker* measures both properties. In case of functions, $f \leq g$ unambiguously means that f is less injective than g. Therefore, $y \ \leq \ x \cdot [\![R]\!]$ above will mean that y is less injective or more defined than x pre-conditioned by $[\![R]\!]$ (a coreflexive). But y is a function and functions are entire (and simple) relations (see the taxonomy of figure 3). In words, R will satisfy FD $x \rightarrow y$ iff attribute y "distinguishes" tuples in R less than attribute x does.

We defer to section 6 the assessment of the calculational advantages of these two pointfree definitions of a functional dependency and move on to PF-transforming definition 2 of a multi-valued dependence (MVD). Recall from above that we have two alternative definitions of MVD, as captured by logical formulæ (17) and (18).

The task of calculating the pointfree transform of (17) is considerably softened by rule (8) given earlier on, which generalizes relational composition. We remind the reader that x and y are attributes which will be regarded as projection functions, as will any combination of (sets of) attributes, eg. xy — which is such that $t[xy] = (t[x], t[y])$. We address the existential quantification of (17) first:

$$
\begin{array}{ll}
& \langle \exists\, t'' \ : \ t'' \in R : \ t[xy] = t''[xy] \ \wedge \ t''[z] = t'[z] \rangle \\
\Leftrightarrow & \{ \ (8) \text{ for } u := (\in R), \text{ and so on } \} \\
& t(ker\, xy \cdot [\![R]\!] \cdot ker\, z) t'
\end{array}
$$

Once we insert this in the overall formula,

$$\langle \forall\, t, t' \ : \ t, t' \in R : \quad t[x] = t'[x] \quad \Rightarrow \quad t(ker\, xy \cdot [\![R]\!] \cdot ker\, z) t' \ \rangle \tag{24}$$

we realize it could be PF-transformed into an instance of relational inclusion (6) should the universal quantifier not be bound to range over tuples t, t' in R. This can be overcome via the following extension to (6): given two binary relations $B \xleftarrow{R,S} A$ and two predicates ψ and ϕ (coreflexively denoted by Ψ and Φ, respectively), then PF-transform rule

$$\langle \forall\, b, a \;:\; (\phi\, b) \;\wedge\; (\psi\, a) :\; b\, R\, a \Rightarrow b\, S\, a \rangle \quad \Leftrightarrow \quad \Phi \cdot R \cdot \Psi \subseteq S \tag{25}$$

holds.

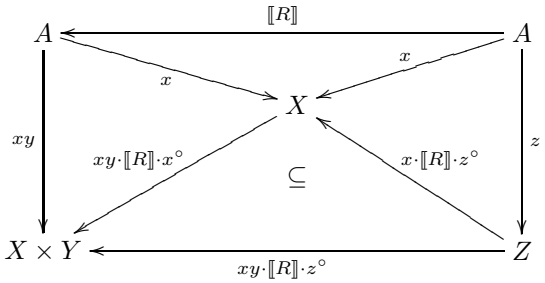
The application of this rule to (24), for $\phi = \psi = (\in R)$, will yield

$$[\![R]\!] \cdot (ker\, x) \cdot [\![R]\!] \quad \subseteq \quad (ker\, xy) \cdot [\![R]\!] \cdot ker\, z$$

which is equivalent to

$$(xy \cdot [\![R]\!] \cdot x^\circ) \cdot (x \cdot [\![R]\!] \cdot z^\circ) \quad \subseteq \quad xy \cdot [\![R]\!] \cdot z^\circ \tag{26}$$

once kernels are expanded via (23) and projection functions are shifted-around via "al-gabr" rules (14, 15) as much as possible. In this way we obtain diagram



(27)

which provides an alternative meaning for MVD-satisfiability: relation R will satisfy $x \twoheadrightarrow y$ iff projection $xy \cdot [\![R]\!] \cdot z^\circ$ "factorizes" through x, for instance

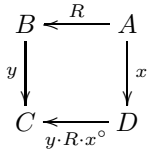
$$\left(\begin{array}{|c|c|c|}\hline & x & y & x \\ \hline t & a & c & a \\ \hline t' & a & c' & a \\ \hline \end{array}\right) \cdot \left(\begin{array}{|c|c|c|}\hline & x & z \\ \hline t & a & b \\ \hline t' & a & b' \\ \hline \end{array}\right) \quad \subseteq \quad \left(\begin{array}{|c|c|c|c|}\hline & x & y & z \\ \hline t & a & c & b \\ \hline t''' & a & c' & b \\ \hline t'' & a & c & b' \\ \hline t' & a & c' & b' \\ \hline \end{array}\right)$$

holding about (19).

5 Two generic (pointfree) definitions

A significant advantage of the allegorical view of relations [16, 10] is the fact that these are *typed*, that is, relational expressions can be *type-checked* and made parametric. So our first efforts go towards generalizing the types of the PF-terms obtained for both FDs and MVDs in the previous section.

Already in diagram (27) nothing is said about types A, X, Y and Z, the implicit interpretation being that A is the type of each tuple in R and that this "includes" types X, Y and Z where attribute x, y and z take values, respectively. But note that diagram (27) still type-checks if $[\![R]\!]$ is replaced by an arbitrary (endo) relation and x, y and z are regarded as arbitrary functions of the types captured by the arrows. The same applies to projection (21), where coreflexive $[\![R]\!]$ can actually be generalized to any binary relation R:



Altogether, we are led to the following PF-transformed (generalized) notions of FD and MVD, where the use of "harpoon" arrows ($\rightharpoonup$) instead of standard arrows is intended to stress the generalization [8]:

Definition 4 *Binary relation* $B \xleftarrow{R} A$ *is said to satisfy the "*$x \rightharpoonup y$*" functional dependency — written* $x \overset{R}{\rightharpoonup} y$ *— iff the* y, x-projection *of* R*, defined*

$$\pi_{y,x} R \quad \overset{\text{def}}{=} \quad y \cdot R \cdot x^\circ \tag{28}$$

is simple (Fig. 3) or, equivalently, iff

$$y \quad \leq \quad x \cdot R^\circ \qquad \textit{holds, cf.} \qquad \begin{array}{ccc} B & \xleftarrow{R} & A \\ {\scriptstyle y}\downarrow & \searrow^{x \cdot R^\circ} & \downarrow{\scriptstyle x} \\ C & & D \end{array} \tag{29}$$

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

Definition 5 *Given endo-relation* $A \xleftarrow{R} A$ *and three functions* $X \xleftarrow{x} A$ *,* $Y \xleftarrow{y} A$ *and* $Z \xleftarrow{z} A$ *, multivalued dependency* $x \overset{R}{\rightharpoonup\!\!\!\!\rightharpoonup}_z y$ *holds (note the subscript* z*) if and only if*

$$R \cdot (\ker x) \cdot R \subseteq (\ker xy) \cdot R \cdot \ker z \tag{30}$$

holds, which is equivalent to

$$xy \cdot R \cdot x^\circ \cdot x \cdot R \cdot z^\circ \quad \subseteq \quad xy \cdot R \cdot z^\circ \tag{31}$$

itself the same as

$$(\pi_{xy,x} R) \cdot (\pi_{x,z} R) \quad \subseteq \quad \pi_{xy,z} R \tag{32}$$

cf. (28). For symmetric R [9]*, (30) can be further re-written into*

$$\ker (x \cdot R^\circ) \subseteq (\ker xy) \cdot R \cdot \ker z \tag{33}$$

As with FDs, x *(resp.* y*) will be referred to as the* antecedent *(resp.* consequent*) of MVD* $x \overset{R}{\rightharpoonup\!\!\!\!\rightharpoonup}_z y$*. Function* z *will be mentioned as the* context [10]*.*
□

[8] See [35] for some technical details left out at this point.
[9] R is symmetric iff $R = R^\circ$. Coreflexives are symmetric.
[10] The context of the standard definition [25] is fixed to $z = S - yx$, that is, z embodies all attributes other than x and y.

6 Calculating with pointfree functional dependencies

Our generalization of FDs to binary relations paves the way to interesting synergies with other notions in mathematics and computing. Among those presented in [35], we pick up the following, which illustrates, in this particular context and in a rather simple way, the role of Russo's dashed lines in fig. 1. The question is: what does it mean for a function h to satisfy FD $f \stackrel{h}{\leftarrow} f$? The (easy) reasoning steps which follow illustrate the "al-gabr" style of (exercise like) calculation which for a moment makes us think we are still at school playing with classical algebra:

$$
\begin{array}{ll}
 & f \stackrel{h}{\leftarrow} f \\
\Leftrightarrow & \quad \{ \text{ (29) } \} \\
 & f \cdot h^\circ \leq f \\
\Leftrightarrow & \quad \{ \text{ (22, 23) and converses } \} \\
 & h \cdot (\mathit{ker}\ f) \cdot h^\circ \subseteq \mathit{ker}\ f \\
\Leftrightarrow & \quad \{ \text{ shift } h, h^\circ \text{ to the right hand side (14, 15) } \} \\
 & \mathit{ker}\ f \subseteq h^\circ \cdot (\mathit{ker}\ f) \cdot h \\
\Leftrightarrow & \quad \{ \text{ go pointwise (10) while renaming } \mathit{ker}\ f \text{ to } \sim_f \ \} \\
 & \langle \forall\ b, a\ ::\ b \sim_f a\ \Rightarrow\ (h\ b) \sim_f (h\ a) \rangle
\end{array}
\tag{34}
$$

The renaming in the last step is intended to make explicit the fact that kernels of functions are equivalence relations [35]. Conversely, any equivalence relation can be regarded as the kernel of a (not unique, in general) function. And (34) can be recognized as the statement that h is *compatible* with $\sim_f$, that is, that $\sim_f$ is a *congruence* with respect to h. As shown in [35], this extends to multiple arguments and heterogeneous compatibility [11], leading us to conclude that functional dependencies *extend congruences relationally*.

Reference [35] also shows the prominent role of the injectivity preorder (22) in FD PF-calculations, in particular in generalizing Armstrong's axioms [25]. An eloquent example is the calculation of axioms F3 (*Additivity* or *Union*) and F4 (*Projectivity*) in a single row, as a linear series of equivalences:

$$
\begin{array}{ll}
 & f \stackrel{R}{\leftarrow} gh \\
\Leftrightarrow & \quad \{ \text{ (29) } \} \\
 & gh \leq f \cdot R^\circ \\
\Leftrightarrow & \quad \{ \text{ see (35) below } \} \\
 & g \leq f \cdot R^\circ\ \wedge\ h \leq f \cdot R^\circ \\
\Leftrightarrow & \quad \{ \text{ (29) twice } \} \\
 & \quad f \stackrel{R}{\leftarrow} g \quad \wedge \quad f \stackrel{R}{\leftarrow} h
\end{array}
$$

The fact assumed in the second step is an instance of Galois connection

$$\langle R, S \rangle \leq T \quad \Leftrightarrow \quad R \leq T\ \wedge\ S \leq T \tag{35}$$

where $\langle R, S \rangle$ denotes the pairing of two relations, recall (10). In the case of functions, $\langle f, g \rangle$ has the same meaning as fg, since $(a, b)\langle f, g \rangle x$ equivales $a = f\ x \wedge b = g\ x$. The interested reader will see in [35] how easy it is to infer (35) and other rules about the $\leq$ preorder from the laws of standard binary relation calculus.

[11] Cf. the Σ-congruences of [19].

PF-transformation makes one aware of the "further structure" of the FD concept which is systematically ignored by the standard theory due to a too narrow view of things. In particular, because antecedents and consequents are viewed as sets of attributes, no further structuring of these is considered apart from union (tupling) and difference. By promoting such attributes to arbitrary functions we can exploit other functional combinators, eg. composition, parameterization, etc.

Below we provide a sample of facts calculated in [35] which exploit such further structure. The first of these

$$x \stackrel{R}{\leftarrow} y \quad \Leftrightarrow \quad f \cdot x \stackrel{R}{\leftarrow} f \cdot y \quad \Leftarrow \quad f \text{ is injective} \tag{36}$$

shows that injective functions are left-cancelable in FDs. Another fact exhibits a close relationship between FDs and (binary) relational projection (28) which enables observer function "trading" between a projection and a (composite) FD:

$$h \stackrel{\pi_{g,f}R}{\leftarrow} k \quad \Leftrightarrow \quad (h \cdot f) \stackrel{R}{\leftarrow} (k \cdot g) \tag{37}$$

A third fact,

$$x \stackrel{R}{\leftarrow} y \quad \Leftrightarrow \quad \mathsf{F}x \stackrel{\mathsf{F}R}{\leftarrow} \mathsf{F}y \tag{38}$$

shows how a parametric type F (relator [6]) can be introduced or removed from a given FD.

Facts (37,38) play not only with antecedents and consequents but also with the target relation R. This cannot be found in full generality in the standard theory because, once again, relations are viewed as sets of tuples manipulated by a limited set of operators (eg. intersection, selection, projection etc). Even a fact as basic as *FD composition*

$$f \stackrel{S \cdot R}{\leftarrow} h \quad \Leftarrow \quad f \stackrel{R}{\leftarrow} g \quad \wedge \quad g \stackrel{S}{\leftarrow} h \tag{39}$$

is absent from the standard theory, despite its being very easy to calculate:

$$\begin{array}{ll} & f \stackrel{R}{\leftarrow} g \quad \wedge \quad g \stackrel{S}{\leftarrow} h \\ \Leftrightarrow & \quad \{ \text{ (29) twice } \} \\ & g \leq f \cdot R^\circ \quad \wedge \quad h \leq g \cdot S^\circ \\ \Rightarrow & \quad \{ \leq\text{-monotonicity of } (\cdot S^\circ) \text{ ; converses } \} \\ & g \cdot S^\circ \leq f \cdot (S \cdot R)^\circ \quad \wedge \quad h \leq g \cdot S^\circ \\ \Rightarrow & \quad \{ \leq\text{-transitivity } \} \\ & h \leq f \cdot (S \cdot R)^\circ \\ \Leftrightarrow & \quad \{ \text{ (29) again } \} \\ & f \stackrel{S \cdot R}{\leftarrow} h \end{array}$$

Note in passing that, together with the obvious

$$f \stackrel{id}{\leftarrow} f \tag{40}$$

FD composition (39) sets up a category whose objects are functions f, g, etc. and whose arrows $f \xrightarrow{R} g$ are relations satisfying $f \stackrel{R}{\leftarrow} g$.

7 Calculating with pointfree multivalued dependencies

MVDs are less intuitive and technically more complex than FDs, a fact already mirrored in the PF-transformation (definition 5) of the MVD concept as given in definition 2.

The main advantage of definition 5 is the freedom given to context z which leads new results or to more general versions of standard results as given in [35]. Our choice in this summary falls on the proof of the theorem of *lossless decomposition* on relations satisfying MVDs, one of the main results of the standard theory [25]:

Theorem 1 *For coreflexive R, MVD $x \overset{R}{\twoheadrightarrow}_z y$ holds* iff *R decomposes losslessly into two relations with schemata xy and xz, respectively:*

$$x \overset{R}{\twoheadrightarrow}_z y \quad \Leftrightarrow \quad (\pi_{y,x}R) \bowtie (\pi_{z,x}R) = \pi_{yz,x}R \tag{41}$$

□

Maier [25] proves this theorem [12] in "implication-first" logic style, in two parts — *if* followed by *only if* — involving existential and universal quantifications over no less than six tuple variables $t, t_1, t_2, t'_1, t'_2, t_3$. The second part involves a proof by contradiction.

The PF proof which follows (taken from [35]) is again another series of equivalences which merges the *if* and *only if* proofs in a single calculation. This is based on the fact that joining two projections which share the same antecedent function, say x, is nothing but binary relation *pairing* (10):

$$(\pi_{y,x}R) \bowtie (\pi_{z,x}R) \quad \overset{\text{def}}{=} \quad \langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle \tag{42}$$

The calculation of (41) goes as follows:

$$\begin{array}{ll}
& (\pi_{y,x}R) \bowtie (\pi_{z,x}R) = \pi_{yz,x}R \\
\Leftrightarrow & \quad \{ \text{ (42) ; (28) three times } \} \\
& \langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle \quad = \quad yz \cdot R \cdot x^\circ \\
\Leftrightarrow & \quad \{ \text{ since } \langle X, Y \rangle \cdot Z \subseteq \langle X \cdot Z, Y \cdot Z \rangle \text{ holds by monotonicity } \} \\
& \langle y \cdot R \cdot x^\circ, z \cdot R \cdot x^\circ \rangle \quad \subseteq \quad yz \cdot R \cdot x^\circ \\
\Leftrightarrow & \quad \{ \text{ "split twist" rule (49) — twice ; converses } \} \\
& \langle y \cdot R \cdot x^\circ, id \rangle \cdot x \cdot R^\circ \cdot z^\circ \quad \subseteq \quad \langle y, x \cdot R^\circ \rangle \cdot z^\circ \\
\Leftrightarrow & \quad \{ \text{ instances of split-fusion — see (50) and (52) in the appendix } \} \\
& \langle y \cdot R \cdot x^\circ, x \cdot x^\circ \rangle \cdot x \cdot R \cdot z^\circ \quad \subseteq \quad \langle y, x \rangle \cdot R \cdot z^\circ \\
\Leftrightarrow & \quad \{ \text{ instances of split-fusion: see (51) and (52) in the appendix } \} \\
& (\langle y, x \rangle \cdot R \cdot x^\circ) \cdot (x \cdot R \cdot z^\circ) \quad \subseteq \quad \langle y, x \rangle \cdot R \cdot z^\circ \\
\Leftrightarrow & \quad \{ \text{ (31) } \} \\
& x \overset{R}{\twoheadrightarrow} y
\end{array}$$

[12] This is Theorem 7.1 in [25].

8 Epilogue

In its original set-theoretic setting, the equivalence between lossless decomposition and MVDs has been known for thirty years. So, what has been gained with its (pointfree) re-statement presented in the current document, after all?

If we compare our calculations to early ways of expressing the same results — see eg. references [15] and [9] — it is clear that in the latter a sheer amount of detail is overlooked in short-circuitous, almost telegram-like proofs, which are trusted on the basis of an almost informal common understanding of naïve set theory.

This includes the use of two, seemingly equivalent, definitions for MVD, one universally quantified over pairs of tuples (17) and the other universally quantified over data values (18) and based on a set-valued function. While the latter is typical of earlier publications in the field (eg. [15, 9]), the former is favoured in textbooks such as [25].

No dedicated proof has been produced — to the best of the author's knowledge — of the equivalence of these two definitions. Below we calculate this equivalence as our last exercise on data dependency theory refactoring, this time involving *transposition*, a device commonly used in the PF-relational calculus:

$$f = \Lambda\ R \quad \Leftrightarrow \quad (bRa \quad \Leftrightarrow \quad b \in f\ a) \tag{43}$$

This establishes the well-known fact that every binary relation R can be converted into a (set-valued) function ΛR, Λ being known as the *power-transpose* isomorphism [10]. This means that *any* set-valued function (eg. Y in definition 3) can be regarded as the *power-transpose* of some binary relation. Substitution $f := \Lambda\ R$ in (43) yields the so-called Λ-cancellation law $b \in (\Lambda R)a \quad \Leftrightarrow \quad bRa$, that is,

$$\in \cdot (\Lambda R) \quad = \quad R \tag{44}$$

which means that $(\Lambda R)a$ yields exactly the set of all b which are related to a by R.

The theory behind relation transposition can be found in eg. [10, 36]. For our purposes below, it is enough to recall from [35] a follow-up of (44),

$$R \cdot T \subseteq S \quad \Leftrightarrow \quad (\Lambda R) \cdot T \quad \subseteq \quad (\Lambda S) \tag{45}$$

and the following generalization to arbitrary x, y and z of rule MVD0 *(Complementation)* of [9]:

$$x \overset{T}{\twoheadrightarrow}_z y \quad \Leftrightarrow \quad x \overset{T}{\twoheadrightarrow}_y z \quad \Leftarrow \quad T \text{ is coreflexive} \tag{46}$$

The proof below of the equivalence between the two given definitions of MVD is, for coreflexive relations, a calculation which re-writes Maier's definition [25] into [9]'s:

$$\begin{array}{ll}
 & x \overset{T}{\twoheadrightarrow}_z y \\
\Leftrightarrow & \quad \{\ (46) \text{ since } T \text{ is coreflexive}\ \} \\
 & x \overset{T}{\twoheadrightarrow}_y z \\
\Leftrightarrow & \quad \{\ (31)\ \} \\
 & y \cdot T \cdot x^\circ \cdot x \cdot T \cdot xz^\circ \quad \subseteq \quad y \cdot T \cdot xz^\circ \\
\Leftrightarrow & \quad \{\ \text{product-cancellation ; } T = T \cdot T^\circ \text{ since } T \text{ is coreflexive}\ \} \\
 & y \cdot T \cdot x^\circ \cdot \pi_1 \cdot xz \cdot T \cdot T^\circ \cdot xz^\circ \quad \subseteq \quad y \cdot T \cdot xz^\circ \\
\Leftrightarrow & \quad \{\ \text{introduce image and power-transpose, cf. (20, 45)}\ \}
\end{array}$$

$$
\begin{array}{ll}
& \Lambda(y \cdot T \cdot x^\circ \cdot \pi_1) \cdot \mathit{img}\,(xz \cdot T) \quad \subseteq \quad \Lambda(y \cdot T \cdot xz^\circ) \\
\Leftrightarrow & \qquad \{ \ \Lambda(R \cdot f) = (\Lambda R) \cdot f \ \text{; then introduce } \lambda_{g,f} T = \Lambda(g \cdot T \cdot f^\circ) \ \} \\
& (\ \lambda_{y,x} T) \cdot \pi_1 \cdot \mathit{img}\,(xz \cdot T) \quad \subseteq \quad \lambda_{y,xz} T \\
\Leftrightarrow & \qquad \{ \ \text{“al-gabr” (14) , since } \lambda_{y,x} T \cdot \pi_1 \text{ is a function} \ \} \\
& \mathit{img}\,(xz \cdot T) \quad \subseteq \quad (\lambda_{y,x} T \cdot \pi_1)^\circ \cdot \lambda_{y,xz} T \\
\Leftrightarrow & \qquad \{ \ \text{go pointwise noting that } xz \cdot T \text{ is simple ; rule } \ b(f^\circ \cdot R \cdot g)a \Leftrightarrow (f\ b) R (g\ a) \ \ (10) \ \} \\
& \langle \forall\ k\ :\ k\ \mathit{img}\,(xz \cdot T)\ k :\ (\lambda_{y,x} T \cdot \pi_1) k = (\lambda_{y,xz} T) k \rangle \\
\Leftrightarrow & \qquad \{ \ \text{rule (48) in the appendix} \ \} \\
& \langle \forall\ k\ :\ \langle \exists\ t\ :\ t \in T :\ (xz\ t) = k \rangle :\ (\lambda_{y,x} T \cdot \pi_1) k = (\lambda_{y,xz} T) k \rangle \\
\Leftrightarrow & \qquad \{ \ \text{rename } k := (b, a) \text{ and simplify} \ \} \\
& \langle \forall\ a, b\ :\ \langle \exists\ t\ :\ t \in T :\ xz\ t = (a, b) \rangle :\ (\lambda_{y,x} T)\ a = (\lambda_{y,xz} T)(a, b) \rangle
\end{array}
$$

We have thus reached (18), the only difference being that function Y generalizes to

$$\lambda_{g,f} = \Lambda \cdot \pi_{g,f} \tag{47}$$

that is, to the power-transpose of projection (28). So, while Y groups y-values only, λ's two parameters cater for any such groups of values:

$$(\lambda_{g,f} R) a = \{ g\ b \mid \langle \exists\ a\ ::\ b\ R\ a\ \wedge\ c = f\ a \rangle \}$$

The reader is referred to [35] for a comprehensive account of (generic) FD and MVD theory in the PF-style, which extends the examples given in this report.

9 Conclusions

> [Symbolisms] *have invariably been introduced to make things easy.* [...] *by the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain.* [...] *Civilisation advances by extending the number of important operations which can be performed without thinking about them.*
>
> Alfred Whitehead (1911)

Pragmatism often leads computer scientists to support the position that, once mathematical evidence of some relevant fact or result is found, that's enough: there is no point in finding "better" ways of providing such evidence, even in the case of clumsy, long-winded, off-putting proofs. Others will think differently: the "better" the proof, the more enduring the result it supports will be. This raises the question: what does "better" mean in this context?

Mathematical evidence is of a social nature: no fact can be regarded as mathematically sound without others checking and acknowledging it. (This is, after all, common to any creative initiative: no work of art can be regarded as such without having a public, for instance.) So, the quality of mathematical evidence is, first of all, measured by the likelihood of others reading and understanding such evidence. Long-winded, awkward-looking proofs with little structure, full of case analyses and "dot-dot-dot" notation are likely to reduce the number of readers able to reach the endpoint of one's arguments.

This is not, however, a simple matter of size, since a mathematical argument cannot be reduced arbitrarily without its loosing substance or changing meaning. How does one then learn how to build *graceful* arguments and formalisms?

Building up on the analogy with art briefly touched above, this question may look as naïve as others like *"How do I learn to write symphonies as graceful as Mozart's?"* Works such as Gasteren's [18], for instance,

provide examples of supple arguments which conquer readers instead of defeating them. After all, such economy between intent and contents has to do with beauty and elegance. Learning how to achieve such a balance may take one's lifetime.

This lecture addresses the topic of effectiveness of formal reasoning in computer science from a different, less philosophical, perspective. The idea is that of mastering (or circumventing) the antagonism between the descriptive and the reasoning sides of computer science notation. Finding the mythical formalism where both co-exist seems difficult in general: description requires verbosity, calculation requires agility. A bridge between the two seems to be the notion of a mathematical *transform* mapping the descriptive notation to the agile notation back and forth.

My field of experimentation in this context has been the application of the *pointfree* (PF) transform to computer science foundations traditionally stated in *pointwise* logics and set theory. The transform consists in removing logic quantification as much as enabled by internalization of logic patterns into combinators of the binary relation calculus. This leads to a kind of "theory re-factoring" in terms of more general and more graceful formulations of the original concepts, leading to easy-to-follow, school-algebra-like proofs.

The choice of target theory in this lecture fell over data dependency theory, the kernel of relational database design. The (apparently odd) decision of explaining n-ary relations in terms of binary relations (contrary to the intuition that *binary* is just a special case of *multi-ary*) is the main ingredient of the re-factoring, followed by the plain application of the PF-transform to descriptive, predicate calculus formulæ, leading to equivalent relational expressions which are calculated with and mapped back to the predicate calculus.

My main conclusion is that data dependency theory would perhaps have been built rather differently should it be based on such a transformation in the first place. It is the complexity of formulæ (16) and (17) that has led database theorists to invest into an axiomatic theory based on inference rules, closures of sets of dependencies, issues such completeness and soundness and so on, instead of calculating *directly* over the definitions themselves, as we have shown it can be done once the definitions are PF-transformed.

The section which follows lists some topics left open by our experiments on PF-transformed data-dependency theory and on the PF-transform itself.

10 Future Work

The proposed change in reasoning style is essential to data-dependence theory refactoring as a whole, in order to meet current standards in software design by calculation [10] and fulfill the original research aims of its pioneers, as expressed in [9] three decades ago: *a general theory that ties together dependencies, relations and operations on relations is still lacking*. Surely, the whole theory has advanced enormously in the thirty years which separate us today from such highly innovative work. However, the intended generic theory has not yet become available because its kernel concepts have remained too specific.

To the best of my knowledge, the research addressed in this lecture is the first comprehensive study of pointfree transformed data dependency theory. But, in a sense, the contribution is more qualitative than quantitative, as much work remains to be done. For instance, other kinds of data dependency (eg. *join* dependencies [25], *difunctional* dependencies [21]) have not been dealt with at all. Besides full coverage of MVD theory and normal forms [25], *null-values* and partial information present another challenge, whereby antecedent and consequent functions of FD/MVDs become partial (ie. pure functions give place to *simple* relations). At first sight, the *Maybe*-transpose of [36] has potential to map this new situation back to the one already dealt with in the current lecture, but semantically things are not that straightforward, as [25] takes some time to explain. Reference [24] provides an interesting update on this problem.

As far as the PF-transform itself is concerned, more experiments have to be carried out before one finds a practical "road map" for "theory PF-refactoring". At this moment, my feeling about the approach is captured by the following guidelines:

- start with *coreflexive* models of the existing theory;

- generalize coreflexives to arbitrary binary relations, "as much as possible" as far as types are concerned;
- fine-tune the generalization by restricting to functions and "seeing what happens" (functions are *one way* relations which help in disambiguating arrow direction).

The prospect of automating the approach is already in the research agenda, see eg. [27]. Last but not least, going further on and formalizing the analogy with the Laplace transform (which so far has only been hinted at) would be a fascinating piece of research in mathematics and computer science in itself, and one which would *put the vast storehouse in order*, to use the words of Lawvere and Schanuel [23], wherefrom one gets firm advice on how to handle analogy in mathematics:

> *Since 1945, when the notion of 'category' was first precisely formulated, these analogies have been sharpened and have become explicit ways in which one subject is transformed into another.*

In these difficult times of widespread pre-scientific software technology, putting the PF transform under the same mathematical umbrella as other integral transforms such as Laplace's would also be rather reassuring in its enhancing the way software sciences are (or could be) framed within engineering mathematics as a whole.

References

[1] C. Aarts, R.C. Backhouse, P. Hoogendijk, E.Voermans, and J. van der Woude. A relational theory of datatypes, December 1992. Available from `www.cs.nott.ac.uk/~rcb`.

[2] T.L. Alves, P.F. Silva, J. Visser, and J.N. Oliveira. Strategic term rewriting and its application to a VDM-SL to SQL conversion. In *FM'05*, volume 3582 of *LNCS*, pages 399–414. Springer-Verlag, 2005.

[3] American National Standards Institute and International Organization for Standardization. *American National Standard programming language COBOL, approved May 10, 1974: ANSI X3.23-1974: revision of X3.23-1968*, volume 21-1 of *Federal Information Processing Standards publication*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, 1974.

[4] R. Backhouse and D. Michaelis. Exercises in quantifier manipulation. In Tarmo Uustalu, editor, *MPC'06, Mathematics of Program Construction 2006*, pages 70–81. Springer Lect. Notes Comp. Sci. (4014), 2006.

[5] R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

[6] R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. In *AMAST'91*, pages 303–362. Springer, 1992.

[7] L.S. Barbosa and J.N. Oliveira. Transposing partial components — an exercise on coalgebraic refinement. *Theoretical Computer Science*, 365(1):2–22, 2006.

[8] L.S. Barbosa, J.N. Oliveira, and A.M. Silva. Calculating invariants as coreflexive bisimulations. In *AMAST'08*, volume 5140 of *LNCS*, pages 83–99. Springer-Verlag, 2008.

[9] Catriel Beeri, Ronald Fagin, and John H. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In D.C.P. Smith, editor, *Proc. 1977 ACM SIGMOD, Toronto*, pages 47–61, 1977.

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

[11] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Longman, Inc., 1999. ISBN 0-201-57168-4.

[12] Peter Pin-Shan Chen. The entity-relationship model: toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[13] A. Cunha, J.N. Oliveira, and J. Visser. Type-safe two-level data transformation. In *FM'06* , volume 4085 of *LNCS*, pages 284–289. Springer-Verlag, Aug. 2006.

[14] H. Doornbos, R. Backhouse, and J. van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1–2):103–135, 1997.

[15] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, 1977.

[16] P.J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.

[17] H. Garcia-Molina, J.D. Ullman, and J.D. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002. ISBN: 0-13-031995-3.

[18] A.J.M. van Gasteren. *On the Shape of Mathematical Arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer, 1990.

[19] J. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of algebraic data types. In Raymond Yeh, editor, *Current Trends in Programming Methodology*, volume IV, pages 80–149. Prentice-Hall, 1978.

[20] ISO. Information technology – database languages – SQL, Nov. 1992. Reference number ISO/IEC 9075:1992(E).

[21] 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.

[22] E. Kreyszig. *Advanced Engineering Mathematics*. J. Wiley & Sons, 6th edition, 1988.

[23] B. Lawvere and S. Schanuel. *Conceptual Mathematics: a First Introduction to Categories*. Cambridge University Press, 1997.

[24] M. Levene and G. Loizou. A generalisation of entity and referential integrity in relational databases. *ITA*, 35(2):113–127, 2001.

[25] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[26] P. Naur and B. Randell, editors. *Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7th to 11th October 1968*. Scientific Affairs Division, NATO, 1969.

[27] C. Necco, J.N. Oliveira, and J. Visser. ESC/PF: Static checking of relational models by calculation, 2008. (in preparation).

[28] Pedro Nunes. *Libro de Algebra en Arithmetica y Geometria*. Original edition by Arnoldo Birckman (Anvers), 1567. Re-edited by the Lisbon Academy of Sciences in 1950 as Vol. VI of Pedro Nunes' Works (Imprensa Nacional de Lisboa).

[29] J.N. Oliveira. "Bagatelle in C arranged for VDM SoLo". *JUCS*, 7(8):754–781, 2001.

[30] J.N. Oliveira. Calculate databases with 'simplicity', September 2004. Presentation at the IFIP WG 2.1 #59 Meeting, Nottingham, UK. (Slides available from the author's website.).

[31] J.N. Oliveira. Functional dependency theory made 'simpler'. Technical Report DI-PURe-05.01.01, DI/CCTC, University of Minho, Gualtar Campus, Braga, 2005. PUReCafé, 2005.01.18 [talk]; available from `http://wiki.di.uminho.pt/twiki/bin/view/Research/PURe/PUReCafe`.

[32] J.N. Oliveira. *Métodos Formais: passado, presente e futuro de um núcleo curricular da Universidade do Minho*, December 2008. Curricular report submitted to the University of Minho under Law 239/2007 dated June 19th, 2007.

[33] J.N. Oliveira. Extended static checking by calculation using the pointfree transform, 2008. Tutorial paper (56 p.) accepted for publication by Springer-Verlag, LNCS series.

[34] J.N. Oliveira. Transforming Data by Calculation. In *GTTSE'07*, volume 5235 of *LNCS*, pages 134–192. Springer, 2008.

[35] J.N. Oliveira. Pointfree foundations for (generic) lossless decomposition, 2008. (Submitted to JACM).

[36] J.N. Oliveira and C.J. Rodrigues. Transposing relations: from *Maybe* functions to hash tables. In *MPC'04*, volume 3125 of *LNCS*, pages 334–356. Springer, 2004.

[37] J.N. Oliveira and C.J. Rodrigues. Pointfree factorization of operation refinement. In *FM'06*, volume 4085 of *LNCS*, pages 236–251. Springer-Verlag, 2006.

[38] O. Ore. Galois connexions, 1944. Trans. Amer. Math. Soc., 55:493-513.

[39] V. Pratt. Origins of the calculus of binary relations. In *Proc. of the 7th Annual IEEE Symp. on Logic in Computer Science*, pages 248–254, Santa Cruz, CA, 1992. IEEE Comp. Soc.

[40] Wolfgang Reisig. *Elements Of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag, September 1998.

[41] L. Russo. *The Forgotten Revolution: How Science Was Born in 300BC and Why It Had to Be Reborn*. Springer-Verlag, September 2003.

[42] P.F. Silva and J.N. Oliveira. 'Galculator': functional prototype of a Galois-connection based proof assistant. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 44–55, New York, NY, USA, 2008. ACM.

[43] P.F. Silva and J.N. Oliveira. Report on the design of a Galculator. Technical Report FAST:08.01, CCTC Research Centre, University of Minho, 2008.

[44] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*. American Mathematical Society, 1987. AMS Colloquium Publications, volume 41, Providence, Rhode Island.

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

[46] Shuling Wang, L.S. Barbosa, and J.N. Oliveira. A Relational Model for Confined Separation Logic. In *TASE 2008*, pages 263–270, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

A Appendix

This appendix quotes from [35] some results of the relational calculus which are referred to in the main text. The first is a specialization of (8), for R a function f and S its converse, whereby one PF-transforms the image of preconditioned $f \cdot \Phi$ (a coreflexive):

$$\begin{array}{lcl} b(f \cdot \Phi \cdot f^\circ)c & \Leftrightarrow & b(img\,(f \cdot \Phi))c \\ & \Leftrightarrow & b = c \wedge \langle \exists\, a \,:\, \phi\, a :\, b = f\, a \rangle \end{array} \tag{48}$$

The following is known in [35] as the "split twist" rule:

$$\langle R, S\rangle \cdot T \subseteq \langle U, V\rangle \cdot X \quad \Leftrightarrow \quad \langle R, T^\circ\rangle \cdot S^\circ \subseteq \langle U, X^\circ\rangle \cdot V^\circ \tag{49}$$

Finally, for simple or coreflexive relations one has:

$$\langle R, T\rangle \cdot S \;=\; \langle R, T \cdot img\, S\rangle \cdot S \quad \Leftarrow \quad S \text{ is simple} \tag{50}$$

$$\langle R, S\rangle \cdot S^\circ = \langle R \cdot S^\circ, img\, S\rangle \quad \Leftarrow \quad S \text{ is simple} \tag{51}$$

$$\langle R, S\rangle \cdot \Phi = \langle R, S \cdot \Phi\rangle \quad \Leftrightarrow \quad \Phi \text{ is coreflexive} \tag{52}$$

The reader is referred to [35] for proofs and further details.