# Pointfree Factorization of Operation Refinement

José N. Oliveira and César J. Rodrigues

Dep. Informática, Universidade do Minho, 4700-320 Braga, Portugal,
{jno,cjr}@di.uminho.pt

**Abstract.** The standard operation refinement ordering is a kind of "meet of opposites": non-determinism reduction suggests "smaller" behaviour while increase of definition suggests "larger" behaviour. Groves' factorization of this ordering into two simpler relations, one per refinement concern, makes it more mathematically tractable but is far from fully exploited in the literature. We present a *pointfree* theory for this factorization which is more agile and calculational than the standard set-theoretic approach. In particular, we show that factorization leads to a simple proof of structural refinement for arbitrary parametric types and exploit factor instantiation across different subclasses of (relational) operation. The prospect of generalizing the factorization to *coalgebraic* refinement is discussed.

**Keywords:** Theoretical foundations ; refinement ; calculation ; reusable theories.

## 1 Introduction

Suppose a component $s$ of some piece of hardware fails and needs to be replaced. Should no exact match be found off the shelf, the maintenance team will have to look around for *compatible* alternatives. What does *compatibility* mean in this context?

Let $r$ be a candidate replacement for $s$ and let the behaviour of both $s$ and $r$ be described by state-transition diagrams indicating, for each state $a$, the set of states reachable from $a$. So both $s$ and $r$ can be regarded as set-valued functions such that, for instance, component $s$ may step from state $a$ to state $b$ iff $b \in (s\ a)$, failing (or behaving unexpectedly) wherever $s\ a$ is the empty set.

The intuition behind $r$ being a safe replacement for $s$ — written $s \vdash r$ — is that not only $r$ should not fail where $s$ does not,

$$\langle \forall\ a\ :\ \emptyset \subset (s\ a) :\ \emptyset \subset (r\ a) \rangle$$

but also that it should behave *"as $s$ does"*. Wherever $(s\ a)$ is nonempty, there is some freedom for $r$ to behave within such a set of choices: $r$ is allowed be more deterministic than $s$. Altogether, one writes

$$s \vdash r \stackrel{\text{def}}{=} \langle \forall\ a\ :\ \emptyset \subset (s\ a) :\ \emptyset \subset (r\ a) \subseteq (s\ a) \rangle \tag{1}$$

This definition of machine compatibility is nothing but a simplified version of that of *operation refinement* [22], the simplification being that one is not spelling out inputs and outputs and that, in general, the two machines $s$ and $r$ above need not share the same state space. This refinement ordering is standard in the discipline of *programming*

*from specifications* [17] and can be found in various guises in the literature — see eg. references [22, 6, 23, 10] among many others. Reference [6] should be singled out for its detailed discussion of the lattice-theoretical properties of the $\vdash$ ordering.

Despite its wide adoption, this definition of $\vdash$ is not free from difficulties. It is a kind of "meet of opposites": non-determinism reduction suggests "smaller" behaviours while increase of definition suggests "larger" behaviours. This "anomaly" makes this standard notion of refinement less mathematically tractable than one would expect. For instance, Groves [10] points out that the principal operators in the Z schema calculus [23] are not monotonic with respect to $\vdash$-refinement [1]. As a way of (partly) overcoming this problem, he puts forward an alternative characterisation of refinement based on the decomposition of $\vdash$ into two simpler relations,

$$s \vdash r \equiv \langle \exists\, t \;::\; s \vdash_{pre} t \;\wedge\; t \vdash_{post} r \rangle \tag{2}$$

one per refinement concern: $\vdash_{pre}$ caters for increasing definition while $\vdash_{post}$ deals with decreasing non-determinism.

The same partition of the refinement relation is addressed in [14], where previous work by Frappier [9] on the $\vdash_{pre}$ ordering is referred to [2]. One of the aims of the current paper is to extend and consolidate the work scattered in [6, 10, 14], where some results are presented without proof and others are supported by either sketchy or convoluted arguments. The idea is to address the subject by reasoning in the pointfree relational calculus which is at the core of the *algebra of programming* [5, 3]. It should be noted that both [6, 10] already use some form of relational notation, somewhat mixed with the Z notation in the case of [10] or interpreted in terms of set-valued functions in [6]. The reasoning, however, is carried out at point-level, either involving predicate logic [10] or set-theory [6].

We follow [14] in resorting to the pointfree relational calculus (which we will refer to as the *pointfree (PF) transform*, see Section 2) all the way through, benefiting from not only its notation economy but also from its elegant reasoning style. It turns out that the theory becomes more general and simpler. Elegant expressions replace lengthy formulæ and easy-to-follow calculations replace pointwise proofs based on case analyses and natural language explanations.

Groves' factorization (2) — which is stated in [14] at PF-level simply by writing

$$\vdash_{pre} \cdot \vdash_{post} = \vdash = \vdash_{post} \cdot \vdash_{pre} \tag{3}$$

— is central to our approach. Thanks to this factorization — which we calculate and justify in a way simpler than in [10] [3] — we are able to justify facts which are stated

[1] According to [10, 8], the literature is scarce in formally approaching this failure of monotonicity, which seems to be well-known among the Z community since the 1980s. See [8] for recent work in the area.

[2] The $\vdash_{pre}/\vdash_{post}$ factorization was suggested around the same time by one of the authors of the current paper [20], but the underlying theory was left unexplored.

[3] No proofs support the factorization in [14], where it is stated in two steps, under the headings: *reduction of nondeterminism commutes with domain extension* and *combination of domain extension and reduction of nondeterminism is refinement*.

but not proved in [6]. Among these, we present a detailed analysis, across the binary relation taxonomy, of the *lattice of specifications* proposed by [6].

As will be explained in the conclusions, this research is part of a broader research initiative aiming at developing a PF-theory for coalgebraic refinement integrating earlier efforts already reported in [16, 4].

*Paper structure.* This paper is laid out as follows. Concerning background, Section 2 provides some motivation on the PF-transform and Section 3 presents an overview of (pointfree) relation algebra. The PF-transformation of (1) is addressed in Section 4. Groves factorization (3) is calculated in sections 5 and 6. Benefits from such a factorization and a proof of structural refinement based on it are presented in Section 7. The paper closes by drawing conclusions which lead to plans for future work.

## 2 On the PF-transform

The main purpose of formal modelling is to identify properties of real-world situations which, once expressed by mathematical formulæ, become abstract models which can be queried and reasoned about. This often raises a kind of *notation* conflict between *descriptiveness* (ie., adequacy to describe domain-specific objects and properties, inc. diagrams or other graphical objects) and *compactness* (as required by algebraic reasoning and solution calculation).

Classical *pointwise* notation in logics involves operators as well as variable symbols, logical connectives, quantifiers, etc. in a way which is hard to scale-up to complex models. This is not, however, the first time this kind of notational conflict arises in mathematics. Elsewhere in physics and engineering, people have learned to overcome it by changing the "mathematical space", for instance by moving (temporarily) from the time-space to the $s$-space in the *Laplace transformation*. Quoting [15], p.242:

> *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****.*

The *pointfree (PF) transform* adopted in this paper is at the heels of this old reasoning technique. Standard set-theory-formulated refinement concepts — such as eg. (1) — are regarded as "hard" problems to be transformed into "simple", *subsidiary equations* dispensing with points and involving only binary relation concepts. 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.

## 3 Overview of the relational calculus

*Relations.* Let $B \xleftarrow{R} A$ denote a binary relation on datatypes $A$ (source) and $B$ (target). We write $bRa$ to mean that pair $(b, a)$ is in $R$. The underlying partial order on relations will be written $R \subseteq S$, meaning that $S$ is either more defined or less deterministic than $R$, that is, $R \subseteq S \equiv bRa \Rightarrow bSa$ for all $a, b$. $R \cup S$ denotes the union of two relations and $\top$ is the largest relation of its type. Its dual is $\bot$, the smallest such relation. Equality on relations can be established by $\subseteq$-antisymmetry: $R = S \equiv R \subseteq S \ \wedge \ S \subseteq R$, or indirect equality: $R = S \equiv \langle \forall \ X \ :: \ X \subseteq R \equiv X \subseteq S \rangle$.

Relations can be combined by three basic operators: composition ($R \cdot S$), converse ($R^\circ$) and meet ($R \cap S$). $R^\circ$ is such that $a(R^\circ)b$ iff $bRa$ holds. Meet corresponds to set-theoretical intersection and composition is defined in the usual way: $b(R \cdot S)c$ holds wherever there exists some mediating $a \in A$ such that $bRa \ \wedge \ aSc$. Everywhere $T = R \cdot S$ holds, the replacement of $T$ by $R \cdot S$ will be referred to as a "factorization" and that of $R \cdot S$ by $T$ as "fusion". (Equation (3) is thus an example of a factorization.) Every relation $B \xleftarrow{R} A$ admits two trivial factorizations, $R = R \cdot id_A$ and $R = id_B \cdot R$ where, for every $X$, $id_X$ is the identity relation mapping every element of $X$ onto itself.

*Coreflexives and orders.* Some standard terminology arises from the $id$ relation: a (endo) relation $A \xleftarrow{R} A$ (often called an *order*) will be referred to as *reflexive* iff $id_A \subseteq R$ holds and as *coreflexive* iff $R \subseteq id_A$ holds. As a rule, subscripts are dropped wherever types are implicit or easy to infer.

Coreflexive relations are fragments of the identity relation which model predicates or sets. The meaning of a *predicate* $p$ is the coreflexive $[\![p]\!]$ such that $b[\![p]\!]a \equiv (b = a) \wedge (p\, a)$, that is, the relation that maps every $a$ which satisfies $p$ (and only such $a$) onto itself. The meaning of a *set* $S \subseteq A$ is $[\![\lambda a.a \in S]\!]$, that is, $b[\![S]\!]a \equiv (b = a) \ \wedge \ a \in S$. Wherever clear from the context, we will omit the $[\![\ ]\!]$ brackets.

Preorders are reflexive, transitive relations, where $R$ is transitive iff $R \cdot R \subseteq R$. Partial orders are anti-symmetric preorders, where $R$ being anti-symmetric means $R \cap R^\circ \subseteq id$. A preorder $R$ is an *equivalence* if it is symmetric, that is, if $R = R^\circ$.

*Taxonomy.* Converse is of paramount importance in establishing a wider taxonomy of binary relations. Let us first define the *kernel* of a relation, $\mathsf{ker}\, R = R^\circ \cdot R$ and its dual, $\mathsf{img}\, R = \mathsf{ker}\, (R^\circ)$, called the *image* of $R$. Since converse commutes with composition, $(R \cdot S)^\circ = S^\circ \cdot R^\circ$ and is involutive, $(R^\circ)^\circ = R$, one has $\mathsf{img}\, R = R \cdot R^\circ$.

Kernel and image lead to the following terminology: a relation $R$ is said to be *entire* (or total) iff its kernel is reflexive; or *simple* (or functional) iff its image is coreflexive. Dually, $R$ is *surjective* iff $R^\circ$ is entire, and $R$ is *injective* iff $R^\circ$ is simple. This terminology is recorded in the following summary table:

| | *Reflexive* | *Coreflexive* |
|---|---|---|
| $\mathsf{ker}\, R$ | entire $R$ | injective $R$ |
| $\mathsf{img}\, R$ | surjective $R$ | simple $R$ |

(4)

A relation is a *function* iff it is both simple and entire. Functions will be denoted by lowercase letters ($f$, $g$, etc.) and are such that $bfa$ means $b = f\ a$. Function converses enjoy a number of properties of which the following is singled out because of its rôle in pointwise-pointfree conversion [2] :

$$b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a) \tag{5}$$

The pointwise definition of kernel of a function $f$, $b(\ker f)a \ \equiv \ f\ b\ =\ f\ a$, stems from (5), whereby it is easy to see that $\top$ is the kernel of every constant function, $1 \xleftarrow{!} A$ included (! is the unique function of its type, where 1 denotes the singleton type).

Isomorphisms are functions which are surjective and injective at the same time. A particular isomorphism is the identity function $id$, which also is the smallest equivalence relation on a particular data domain. So, $b\ id\ a$ means the same as $b = a$.

*Functions and relations.* The interplay between functions and relations is a rich part of the binary relation calculus. In particular, given two preorders $\leq$ and $\sqsubseteq$, one may relate arguments and results of pairs of functions $f$ and $g$ in, essentially, two ways:

$$f \cdot \sqsubseteq \subseteq \leq \cdot g \tag{6}$$

$$f^\circ \cdot \sqsubseteq = \leq \cdot g \tag{7}$$

As we shall see shortly, (6) is equivalent to $\sqsubseteq \subseteq f^\circ \cdot \leq \cdot g$. For $f = g$, this establishes $\sqsubseteq$ to $\leq$ monotonicity, thanks to (5). Both $f, g$ in the other case (7) are monotone and said to be *Galois connected*, $f$ (resp. $g$) being referred to as the *lower* (resp. *upper*) adjoint of the connection. By introducing variables in both sides of (6) via (5), we obtain

$$(f\ b) \sqsubseteq a \ \equiv \ b \leq (g\ a) \tag{8}$$

For further details on the rich theory of Galois connections and examples of application see [1, 2]. Galois connections in which the two preorders are relation inclusion ($\leq, \sqsubseteq := \subseteq, \subseteq$) are particularly interesting because the two adjoints are relational combinators and the connection itself is their universal property. The following table lists connections which are relevant for this paper:

| $(f\ X) \subseteq Y \equiv X \subseteq (g\ Y)$ | | | |
|---|---|---|---|
| **Description** | $f$ | $g$ | **Obs.** |
| Converse | $(\_)^\circ$ | $(\_)^\circ$ | |
| *Shunting* rule | $(f\cdot)$ | $(f^\circ\cdot)$ | NB: $f$ is a function |
| "Converse" *shunting* rule | $(\cdot f^\circ)$ | $(\cdot f)$ | NB: $f$ is a function |
| Left-division | $(R\cdot)$ | $(R \setminus\ )$ | read "$R$ under ..." |
| Right-division | $(\cdot R)$ | $(\ /\ R)$ | read "...over $R$" |
| range | $\rho$ | $(\cdot \top)$ | lower $\subseteq$ restricted to coreflexives |
| domain | $\delta$ | $(\top \cdot)$ | lower $\subseteq$ restricted to coreflexives |

(9)

The connection associated with the *domain* operator will be particularly useful later on, whereby we infer that it is monotonic and commutes with join

$$\delta\,(R \cup S) = (\delta\,R) \cup (\delta\,S) \tag{10}$$

(as all lower-adjoints do [4]) and can be switched to so-called *conditions* [12]

$$\delta\, R \subseteq \delta\, S \equiv\, !\cdot R \subseteq\, !\cdot S \tag{11}$$

wherever required, since $\top = \mathsf{ker}\, !$.

Left-division is another relational combinator relevant for this paper, from whose connection in (9) not only the following pointwise definition can be inferred [3],

$$b\,(R \setminus Y)\,a \equiv \langle \forall\, c\ :\ c\,R\,b :\ c\,Y\,a\rangle \tag{12}$$

but also the following properties which will be useful in the sequel, for $\Phi$ coreflexive:

$$(R \cup T) \setminus S = (R \setminus S) \cap (T \setminus S) \tag{13}$$

$$(R \cdot \Phi \setminus S) \cap \Phi = (R \setminus S) \cap \Phi \tag{14}$$

## 4 Warming up

According to the PF-transformation strategy announced in Section 2, our first task will be to PF-transform (1). We first concentrate on transforming the test for non-failure states, which occurs twice in the formula, $(s\ a) \supset \emptyset$ and $(r\ a) \supset \emptyset$. A set is nonempty iff it contains at least one element. Therefore,

$$\begin{array}{ll}
(s\ a) \supset \emptyset \equiv & \langle \exists\, x\ ::\ x \in (s\ a)\rangle \\
\equiv & \quad \{\ \text{idempotence of}\ \wedge\ \} \\
& \langle \exists\, x\ ::\ x \in (s\ a)\ \wedge\ x \in (s\ a)\rangle \\
\equiv & \quad \{\ \text{(5) twice and converse}\ \} \\
& \langle \exists\, x\ ::\ a(\in \cdot s)^{\circ} x\ \wedge\ x(\in \cdot s)a\rangle \\
\equiv & \quad \{\ \text{introduce}\ b = a\ \text{; composition}\ \} \\
& b = a\ \wedge\ b((\in \cdot s)^{\circ} \cdot (\in \cdot s))a \\
\equiv & \quad \{\ \text{introduce kernel}\ \} \\
& b = a\ \wedge\ b(\mathsf{ker}\,(\in \cdot s))a
\end{array}$$

Then we address the whole formula:

$$\begin{array}{ll}
& s \vdash r \\
\equiv & \quad \{\ (1)\ \} \\
& \langle \forall\, a\ :\ (s\ a) \supset \emptyset :\ \emptyset \subset (r\ a) \subseteq (s\ a)\rangle
\end{array}$$

[4] All $f$ and $g$ are monotonic by definition, as Galois adjoints. Moreover, the $f$s commute with join and the $g$s with meet. Thus we obtain monotonicity and (10) for free, whose proof as law 3.2 in [10] is unnecessary. It should be mentioned that some rules in table (9) appear in the literature under different guises and usually not identified as Galois connections. For instance, the *shunting* rule is called *cancellation law* in [23].

$$
\begin{array}{ll}
\equiv & \{ \text{ expand } \emptyset \subset (r\ a) \subseteq (s\ a)\ \} \\
& \langle \forall\ a\ :\ (s\ a) \supset \emptyset :\ \emptyset \subset (r\ a)\ \wedge\ (r\ a) \subseteq (s\ a) \rangle \\
\equiv & \{ \text{ expand tests for non-failure state and replace } (r\ a) \text{ by } (r\ b), \text{ cf. } b = a\ \} \\
& \langle \forall\ a, b\ :\ b = a\ \wedge\ b(\mathsf{ker}\ (\in \cdot\ s))a :\ b = a\ \wedge\ b(\mathsf{ker}\ (\in \cdot\ r))a\ \wedge\ (r\ b) \subseteq (s\ a) \rangle \\
\equiv & \{ \ \delta\ R = \mathsf{ker}\ R \cap id \text{ is a closed formula for the domain operator [5, 3]}\ \} \\
& \langle \forall\ a, b\ :\ b(\delta\ (\in \cdot\ s))a :\ b(\delta\ (\in \cdot\ r))a\ \wedge\ (r\ b) \subseteq (s\ a) \rangle \\
\equiv & \{ \text{ expand set-theoretic inclusion } \} \\
& \langle \forall\ a, b\ :\ b(\delta\ (\in \cdot\ s))a :\ b(\delta\ (\in \cdot\ r))a\ \wedge\ \langle \forall\ c\ :\ c \in (r\ b) :\ c \in (s\ b) \rangle \rangle \\
\equiv & \{ \text{ (5) twice ; then introduce left-division (12) } \} \\
& \langle \forall\ a, b\ :\ b(\delta\ (\in \cdot\ s))a :\ b(\delta\ (\in \cdot\ r))a\ \wedge\ b((\in \cdot\ r) \setminus (\in \cdot\ s))a \rangle \\
\equiv & \{ \text{ remove points ; relational inclusion and meet } \} \\
& \delta\ (\in \cdot\ s) \subseteq \delta\ (\in \cdot\ r) \cap ((\in \cdot\ r) \setminus (\in \cdot\ s)) \\
\equiv & \{ \text{ remove membership by defining } R = \in \cdot\ r \text{ and } S = \in \cdot\ S\ \} \\
& \delta\ S \subseteq \delta\ R \cap (R \setminus S)
\end{array}
$$

Function $s$ (resp. $r$) can be identified with the *power-transpose* [5, 19] of binary relation $S$ (resp. $R$). Since transposition is an isomorphism, we can safely lift our original ordering on set-valued state-transition functions to state-transition relations and establish the relational PF-transform of (1) as follows:

$$S \vdash R \ \equiv\ \delta\ S \subseteq (R \setminus S) \cap \delta\ R \tag{15}$$

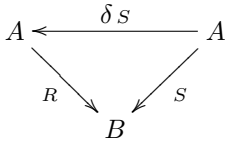
which converts to

$$S \vdash R \ \equiv\ (\delta\ S \subseteq \delta\ R)\ \wedge\ (R \cdot \delta\ S \subseteq S) \tag{16}$$

once Galois connections of meet and left-division (9) are taken into account.

Most definitions of the refinement ordering in the literature — eg. [22, 6, 23, 10] — are pointwise variants of (16). The calculations above show these to be equivalent to our starting version (1), which instantiates a "coalgebraic pattern" favoured in automata theory and coalgebraic refinement [16].

It is easy to see that the target types of both $S, R$ in (16) need not be the same:




So, PF-transformed $S \vdash R$ covers other refinement situations, namely that of an *implicit specification* [13] $S$ being refined by some function $f$,

$$S \vdash f \ \equiv\ \delta\ S \subseteq f^{\circ} \cdot S$$

whereby — back to points and thanks to (5) — we obtain, in classical "VDM-speak"

$$\forall a.\ \text{pre-}S(a) \Rightarrow \text{post-}S(f\ a, a)$$

which is nothing but the *implicit function specification* proof-rule given by [13].

It is in this (wider) context that the $\vdash$ ordering is presented in [6], where it is called the *less-defined* relation on specifications and is shown to be a semi-lattice universally lower-bounded by the empty specification $\bot$. The proof that it is a partial order is telegram-like in the paper. By contrast, the existence of a greatest lower bound (*glb*) is the subject of a proposition proved in typical *invent & verify* style — a *glb* definition is guessed first, which is then shown to be a lower bound and finally proved to be maximal among all lower bounds.

To illustrate the shift from *verification* to *calculation* brought forth by the PF-transform, we will *calculate* the *glb* of $\vdash$ (denoted $\sqcap$) as the (unique) solution to universal property

$$X \vdash R \sqcap S \quad \equiv \quad X \vdash R \ \wedge\ X \vdash S \tag{17}$$

Let us solve this equation for unknown $\sqcap$:

$$\begin{array}{ll}
& X \vdash R \sqcap S \\
\equiv & \quad \{\ (17)\ \} \\
& X \vdash R\ \wedge\ X \vdash S \\
\equiv & \quad \{\ (15) \text{ twice; composition of coreflexives is intersection}\ \} \\
& \delta\, X \subseteq ((R \setminus X \cap S \setminus X)) \cap \delta\, R \cdot \delta\, S \\
\equiv & \quad \{\ (13)\ \} \\
& \delta\, X \subseteq (R \cup S) \setminus X \cap \delta\, R \cdot \delta\, S \\
\equiv & \quad \{\ (14) \text{ for } R, S, \Phi := R \cup S, X, \delta\, R \cdot \delta\, S\ \} \\
& \delta\, X \subseteq (((R \cup S) \cdot \delta\, R \cdot \delta\, S) \setminus X) \cap (\delta\, R \cdot \delta\, S) \\
\equiv & \quad \{\ \delta\,((R \cup S) \cdot \delta\, R \cdot \delta\, S) = \delta\, R \cdot \delta\, S \text{ (coreflexives) ; (15)}\ \} \\
& X \vdash ((R \cup S) \cdot \delta\, R \cdot \delta\, S) \\
:: & \quad \{\ \text{indirect equality (Section 3) on partial order} \vdash [1, 3]\ \} \\
& R \sqcap S = (R \cup S) \cdot \delta\, R \cdot \delta\, S
\end{array}$$

Thus we have deduced

$$R \sqcap S = (R \cup S) \cdot \delta\, R \cdot \delta\, S \tag{18}$$

which, back to points (1), will look like

$$(r \sqcap s)a \quad \equiv \quad if\ \ (r\ a) = \emptyset \vee (s\ a) = \emptyset\ \ then\ \ \emptyset\ \ else\ \ (r\ a) \cup (s\ a) \tag{19}$$

where $r$ (resp. $s$) is the power-transpose of $R$ (resp. $S$). (The reader is invited to calculate (19) as solution to (17) by directly resorting to pointwise $\vdash$ (1) instead of (15).)

## 5 Refinement sub-relations

Recall the two conjuncts of (16), $\delta\, S \subseteq \delta\, R$ and $R \cdot \delta\, S \subseteq S$. Groves [10] freezes the former in defining a sub-relation $\vdash_{post}$ of $\vdash$,

$$S \vdash_{post} R \;\equiv\; S \vdash R \;\wedge\; \delta\, R \subseteq \delta\, S \tag{20}$$

where extra clause $\delta\, R \subseteq \delta\, S$ prevents definition increase (by antisymmetry). Similarly, he puts forward another sub-relation $\vdash_{pre}$ of $\vdash$,

$$S \vdash_{pre} R \;\equiv\; S \vdash R \;\wedge\; S \subseteq R \cdot \delta\, S \tag{21}$$

where extra clause $S \subseteq R \cdot \delta\, S$ prevents from increasing determinacy.

How useful are these sub-orderings? We will devote the remainder of the paper to exploiting the underlying theory and showing them to be useful beyond their original context of definition [10]. First of all, facts

$$\vdash_{pre} \subseteq \vdash \;,\; \vdash_{post} \subseteq \vdash \tag{22}$$

follow immediately from the definitions (20,21) above. That both $\vdash_{pre}$ and $\vdash_{post}$ can be expressed independently of $\vdash$ is simple to calculate, first for $\vdash_{pre}$,

$$\begin{array}{ll}
& S \vdash_{pre} R \\
\equiv & \quad \{\ \text{(21) and (16) ; antisymmetry}\ \} \\
& R \cdot \delta\, S = S \;\wedge\; \delta\, S \subseteq \delta\, R \\
\equiv & \quad \{\ \text{switch to conditions (11)}\ \} \\
& R \cdot \delta\, S = S \;\wedge\; !\cdot S \subseteq\, !\cdot R \\
\equiv & \quad \{\ \text{substitution of } S \text{ by } R \cdot \delta\, S\ \} \\
& R \cdot \delta\, S = S \;\wedge\; !\cdot R \cdot \delta\, S \subseteq\, !\cdot R \\
\equiv & \quad \{\ \delta\, S \text{ is coreflexive } (\delta\, S \subseteq id) \text{ ; monotonicity of composition}\ \} \\
& R \cdot \delta\, S = S \;\wedge\; \textsc{True} \\
\equiv & \quad \{\ \text{trivia}\ \} \\
& R \cdot \delta\, S = S
\end{array}$$

and then for $\vdash_{post}$:

$$\begin{array}{ll}
& S \vdash_{post} R \\
\equiv & \quad \{\ \text{(20) and (16)}\ \} \\
& R \cdot \delta\, S \subseteq S \;\wedge\; \delta\, R = \delta\, S \\
\equiv & \quad \{\ \text{substitution of } \delta\, S \text{ by } \delta\, R\ \} \\
& R \cdot \delta\, R \subseteq S \;\wedge\; \delta\, R = \delta\, S \\
\equiv & \quad \{\ R \cdot \delta\, R = R\ \} \\
& R \subseteq S \;\wedge\; \delta\, R = \delta\, S
\end{array}$$

Let us record these results, which are the PF-counterparts to laws 4.3 and 4.4 in [10], respectively,

$$S \vdash_{pre} R \;\equiv\; R \cdot \delta\, S = S \tag{23}$$
$$S \vdash_{post} R \;\equiv\; R \subseteq S \;\wedge\; \delta\, R = \delta\, S \tag{24}$$

noting that (24) can be written in less symbols as PF-equality

$$\vdash_{post} \;=\; \subseteq^{\circ} \cap \ker \delta \tag{25}$$

Thus, by definition, $\vdash_{post}$ is a partial order, since the meet of a partial order ($\subseteq^{\circ}$) with an equivalence ($\ker \delta$) is a partial order. (The proof that $\vdash_{pre}$ is also a partial order is elementary, see eg. [21].)

What does it mean to impose $\vdash_{pre}$ and $\vdash_{post}$ at the same time? We calculate:

$$\begin{array}{ll} & S \vdash_{pre} R \;\wedge\; S \vdash_{post} R \\ \equiv & \quad \{\ (23), (24)\ \} \\ & R \cdot \delta\, S = S \;\wedge\; \delta\, S = \delta\, R \;\wedge\; R \subseteq S \\ \equiv & \quad \{\ \text{substitution of } \delta\, S \text{ by } \delta\, R\ \} \\ & R \cdot \delta\, R = S \;\wedge\; \delta\, R = \delta\, R \;\wedge\; R \subseteq S \\ \equiv & \quad \{\ \text{property } R = R \cdot \delta\, R\ \} \\ & R = S \;\wedge\; R \subseteq S \\ \equiv & \quad \{\ R = S \Rightarrow R \subseteq S\ \} \\ & R = S \end{array}$$

This result (law 4.7 in [10]) PF-transforms to $\vdash_{pre} \cap \vdash_{post} = id$, whose "antisymmetric pattern" captures the opposition between the components $\vdash_{pre}$ and $\vdash_{post}$ of $\vdash$: to increase determinism only *and* definition only at the same time is contradictory. This relative antisymmetry between $\vdash_{pre}$ and $\vdash_{post}$ can also be inferred from facts

$$S \vdash_{post} R \Rightarrow R \subseteq S \tag{26}$$
$$S \vdash_{pre} R \Rightarrow S \subseteq R \tag{27}$$

the former arising immediately from (24) and the latter holding by transitivity: $S \vdash_{pre} R$ implies $S \subseteq R \cdot \delta\, S$ and $R \cdot \delta\, S \subseteq R$ holds.

## 6 Factorization of the refinement relation

We proceed to showing that the sequential composition of subrelations $\vdash_{pre}$ and $\vdash_{post}$ is — in any order — the refinement relation $\vdash$ itself. As we shall briefly see, this is where our calculational style differs more substantially from that of [10].

That $\vdash_{pre}$ and $\vdash_{post}$ are *factors* of $\vdash$ — that is, $\vdash_{post} \cdot \vdash_{pre} \subseteq \vdash$ and $\vdash_{pre} \cdot \vdash_{post} \subseteq \vdash$ — is obvious, recall (22) and composition monotonicity. So we are left with facts

$$\vdash \subseteq \vdash_{pre} \cdot \vdash_{post} \tag{28}$$
$$\vdash \subseteq \vdash_{post} \cdot \vdash_{pre} \tag{29}$$

to prove. As earlier on, instead of postulating the decompositions and then proving them, we will calculate (deduce) them. Two auxiliary results will be required:

$$S \vdash_{post} S \cap R \;\equiv\; \delta\, S = \delta\,(R \cap S) \tag{30}$$
$$S \vdash_{pre} S \cup R \;\equiv\; R \cdot \delta\, S \subseteq S \tag{31}$$

The proof of (30) immediate from the definition of $\vdash_{post}$ (24). That of (31) follows:

$$\begin{array}{ll} & S \vdash_{pre} S \cup R \\ \equiv & \quad \{ \text{ definition of } \vdash_{pre} \} \\ & (S \cup R) \cdot \delta\, S = S \\ \equiv & \quad \{ \; (\cdot \delta\, S) \text{ is a lower adjoint (9) } \} \\ & (S \cdot \delta\, S) \cup (R \cdot \delta\, S) = S \\ \equiv & \quad \{ \; S \cdot \delta\, S = S \; \} \\ & S \cup R \cdot \delta\, S = S \\ \equiv & \quad \{ \; A \cup B = B \equiv A \subseteq B \; \} \\ & R \cdot \delta\, S \subseteq S \end{array}$$

We are now ready to calculate (28):

$$\begin{array}{ll} & S \vdash R \\ \equiv & \quad \{ \; (16) \; \} \\ & R \cdot \delta\, S \subseteq S \;\wedge\; \delta\, S \subseteq \delta\, R \\ \equiv & \quad \{ \; A \cup B = B \equiv A \subseteq B \; \} \\ & R \cdot \delta\, S \subseteq S \;\wedge\; (\delta\, S) \cup (\delta\, R) = \delta\, R \\ \equiv & \quad \{ \; (10) \; \} \\ & R \cdot \delta\, S \subseteq S \;\wedge\; \delta\,(S \cup R) = \delta\, R \\ \equiv & \quad \{ \; (30), \text{ since } R = R \cap (S \cup R) \; \} \\ & R \cdot \delta\, S \subseteq S \;\wedge\; (S \cup R) \vdash_{post} R \cap (S \cup R) \\ \equiv & \quad \{ \; (31) \text{ and } R = R \cap (S \cup R) \; \} \\ & (S \vdash_{pre} S \cup R) \;\wedge\; (S \cup R) \vdash_{post} R \\ \Rightarrow & \quad \{ \; \text{logic} \; \} \end{array}$$

$$
\begin{array}{ll}
& \langle \exists\, T \;::\; S \vdash_{pre} T \;\wedge\; T \vdash_{post} R \rangle \\
\equiv & \qquad \{ \text{ composition } \} \\
& S(\vdash_{pre} \cdot \vdash_{post}) R
\end{array}
$$

Concerning (29):

$$
\begin{array}{ll}
& S \vdash R \\
\equiv & \qquad \{ \text{ since } S \vdash R \Rightarrow \delta\, S = \delta\,(S \cap R) \text{ [21]; (16) } \} \\
& \delta\, S = \delta\,(S \cap R) \;\wedge\; R \cdot \delta\, S \subseteq S \\
\equiv & \qquad \{ \cap\text{-universal and } \delta\, S \text{ is coreflexive } \} \\
& \delta\, S = \delta\,(S \cap R) \;\wedge\; R \cdot \delta\, S \subseteq S \cap R \\
\equiv & \qquad \{ \text{ substitution } \} \\
& \delta\, S = \delta\,(S \cap R) \;\wedge\; R \cdot \delta\,(S \cap R) \subseteq S \cap R \\
\equiv & \qquad \{ \text{ (31) } \} \\
& \delta\, S = \delta\,(S \cap R) \;\wedge\; (S \cap R) \vdash_{pre} (S \cap R) \cup R \\
\equiv & \qquad \{ \text{ (30) and } S \cap R \subseteq R \; \} \\
& (S \vdash_{post} S \cap R) \;\wedge\; (S \cap R) \vdash_{pre} R \\
\Rightarrow & \qquad \{ \text{ logic } \} \\
& \langle \exists\, T \;::\; S \vdash_{post} T \;\wedge\; T \vdash_{pre} R \rangle \\
\equiv & \qquad \{ \text{ composition } \} \\
& S(\vdash_{post} \cdot \vdash_{pre}) R
\end{array}
$$

In summary, we have the two alternative ways to factor the refinement relation announced in (3). This embodies laws 4.8 and 4.9 of [10], where they are proved in first-order logic requiring negation and consistency [5]. These requirements, which have no counterpart in our calculations above, should be regarded as spurious.

## 7 Taking advantage of the factorization

Factorizations such as that given by (3) are very useful in mathematics in general. For our purposes, the rôle of (3) is three-fold. On the one hand, properties of the composition — eg. transitivity, reflexivity — can be easily inferred from similar properties of factors $\vdash_{pre}$ and $\vdash_{post}$ [21]. On the other hand, one can look for results with hold for the individual factors $\vdash_{pre}$ and/or $\vdash_{post}$ and do not hold (in general) for $\vdash$. For instance,

[5] In [10, 6], two relations $R$ and $S$ are regarded as *consistent* iff $\delta\,(R \cap S) = (\delta\, R) \cap (\delta\, S)$ holds.

meet ($R \cap S$) is $\vdash_{pre}$-monotonic but not $\vdash$-monotonic (law 5.1 in [10]). This aspect of the factorization is of practical value and in fact the main motivation in [10]: complex refinement steps can be factored in *less big a gap* ones involving only one factor $\vdash_{pre}$ (resp. $\vdash_{post}$) and $\vdash_{pre}$ (resp. $\vdash_{post}$) monotonic operators.

Space restraints prevent us from presenting our calculation of monotonicity laws 5.1 and 5.4 of [10], respectively

$$S \vdash_{pre} R \ \wedge \ T \vdash_{pre} U \ \Rightarrow \ S \cap T \vdash_{pre} R \cap U \tag{32}$$

$$S \vdash_{post} R \ \wedge \ T \vdash_{post} U \ \Rightarrow \ S \cup T \vdash_{post} R \cup U \tag{33}$$

which the interested reader will find in [21]. We anticipate that, unlike [10], pointfree calculation doesn't require negation.

Last but not least, there is another practical outcome of factorization (3) which was left unexploited in [10]: the fact that it makes it easy to analyse the (semi-)lattice of operations ordered by $\vdash$ [6], in particular concerning the behaviour of factors $\vdash_{pre}$ and $\vdash_{post}$ for some of the relation subclasses studied in Section 3. For instance, if by construction one knows that the operation under refinement is *simple* (vulg. a partial function), one can safely replace $\vdash$ by the appropriate factors tabulated in

| Binary relation sub-class | $\vdash_{post}$ | $\vdash_{pre}$ | $\vdash$ | |
|---|---|---|---|---|
| Entire relations | $\subseteq^{\circ}$ | $id$ | $\subseteq^{\circ}$ | (a) |
| Simple relations | $id$ | $\subseteq$ | $\subseteq$ | (b) |
| Functions | $id$ | $id$ | $id$ | (c) |

(34)

Let us justify (34): $\vdash_{pre} = id$ in case (34a) follows directly from (23), in which case equation (3) yields $\vdash = \vdash_{post}$. Moreover, $\vdash_{post} = \subseteq^{\circ}$ holds since domain ($\delta$) is a constant function within the class of *entire* relations and thus $\mathsf{ker}\, \delta = \top$ in (25). The proof of (34b) is immediate in the case of $\vdash_{post} = id$, since (24) restricted to simple relations establishes equality at once. Concerning $\vdash_{pre} = \subseteq$, our calculation to follow will rely on relaxing function $f$ to a simple relation $S$ in the *shunting* rules in (9), leading to rules [18]

$$S \cdot R \subseteq T \ \equiv \ (\delta\, S) \cdot R \subseteq S^{\circ} \cdot T \tag{35}$$

$$R \cdot S^{\circ} \subseteq T \ \equiv \ R \cdot \delta\, S \subseteq T \cdot S \tag{36}$$

which, however, are not Galois connections. We reason:

$$\begin{array}{ll} & S \vdash_{pre} R \\ \equiv & \quad \{ \text{ (23) ; anti-symmetry } \} \\ & R \cdot \delta\, S \subseteq S \ \wedge \ S \subseteq R \cdot \delta\, S \\ \equiv & \quad \{ \text{ shunt on simple } R \text{ (35) and } S \text{ (36) ; } S = S \cdot \delta\, S \ \} \\ & \delta\, R \cdot S^{\circ} \subseteq R^{\circ} \ \wedge \ S \cdot \delta\, S \subseteq R \cdot \delta\, S \\ \equiv & \quad \{ \text{ converses } \} \\ & S \cdot \delta\, R \subseteq R \ \wedge \ S \cdot \delta\, S \subseteq R \cdot \delta\, S \end{array}$$

$$
\begin{array}{ll}
\equiv & \{\ \delta\, S \subseteq \delta\, R \text{, cf. (27) and monotonicity of } \delta\ \} \\
& S \subseteq R\ \wedge\ S \cdot \delta\, S \subseteq R \cdot \delta\, S \\
\equiv & \{\ \text{first conjunct implies the second (monotonicity)}\ \} \\
& S \subseteq R
\end{array}
$$

Finally, (34c) follows from functions being entire and simple at the same time.

A comment on the *glb* of $\vdash_{pre}$ restricted to simple relations, ie. deterministic but possibly failing operations (partial functions): pointfree calculation yields $R \sqcap S = R \cap S$ in this case, which agrees with $\subseteq$ in (34b) but contrasts to factor $R \cup S$ in (18). It can be easily calculated that simplicity of $R \sqcap S$ (18) is equivalent to both $R, S$ being simple and $R \cdot S^{\circ} \subseteq id$, which is equivalent, thanks to (36), to $R \cdot \delta\, S \subseteq S$, itself equivalent to $S \cdot \delta\, R \subseteq R$. From these we calculate $(R \cup S) \cdot \delta\, R \cdot \delta\, S \subseteq R \cap S$. Since $R \cap S \subseteq (R \cup S) \cdot \delta\, R \cdot \delta\, S$, we obtain $\sqcap = \cap$ for simple relations.

*Structural refinement.* We close the technical part of the paper by presenting a law which is particularly useful in modular (structural) refinement and whose proof relies heavily on factorization (3):

$$S \vdash R \ \Rightarrow\ \mathsf{F}\, S \vdash \mathsf{F}\, R \tag{37}$$

This law expresses $\vdash$-monotonicity of an arbitrary parametric type $\mathsf{F}$. Technically, the parametricity of $\mathsf{F}$ is captured by regarding it as a *relator* [1, 5], a concept which extends *functors* to relations: $\mathsf{F}\, A$ describes a parametric type while $\mathsf{F}\, R$ is a relation from $\mathsf{F}\, A$ to $\mathsf{F}\, B$ provided $R$ is a relation from $A$ to $B$. Relators are monotonic and commute with composition, converse and the identity.

Fact (37) is another example of a property of operation refinement whose proof uses the strategy of promoting $\vdash_{post}/\vdash_{pre}$ properties to $\vdash$. We need the auxiliary result that every relator $\mathsf{F}$ is both $\vdash_{pre}/\vdash_{post}$-monotonic:

$$\mathsf{F} \cdot \vdash_{post} \subseteq \vdash_{post} \cdot \mathsf{F} \tag{38}$$

$$\mathsf{F} \cdot \vdash_{pre} \subseteq \vdash_{pre} \cdot \mathsf{F} \tag{39}$$

The PF-calculations which support (38,39) are omitted for space economy and can be found in [21]. Then the calculation of (37) is an easy task:

$$
\begin{array}{ll}
& \textsc{True} \\
\equiv & \{\ (38)\ \} \\
& \mathsf{F} \cdot \vdash_{post} \subseteq \vdash_{post} \cdot \mathsf{F} \\
\Rightarrow & \{\ \text{monotonicity of composition}\ \} \\
& \mathsf{F} \cdot \vdash_{post} \cdot \vdash_{pre} \subseteq \vdash_{post} \cdot \mathsf{F} \cdot \vdash_{pre} \\
\Rightarrow & \{\ (39) \text{ and } \subseteq\text{-transitivity}\ \} \\
& \mathsf{F} \cdot \vdash_{post} \cdot \vdash_{pre} \subseteq \vdash_{post} \cdot \vdash_{pre} \cdot \mathsf{F}
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{\ (3)\ \} \\
& \mathsf{F} \cdot \vdash \subseteq \vdash \cdot \mathsf{F} \\
\equiv & \{\ \text{shunt over } \mathsf{F}\ (9) \text{ and then go pointwise on } S \text{ and } R\ \} \\
& R \vdash S \Rightarrow \mathsf{F}\, R \vdash \mathsf{F}\, S
\end{array}
$$

## 8 Conclusions and future work

Refinement is among the most studied topics in software design theory. An extensive treatment of the subject can be found in [7]. It is, however, far from being an easy-to-use body of knowledge, a remark which is mirrored on terminology — cf. *downward*, *upward* refinement [11], *forwards*, *backwards* refinement [11, 23, 16], *S,SP,SC*-refinement [8] and so on.

Boudriga *et al* [6] refer prosaically to the refinement ordering (denoted $\vdash$ in the current paper) as the *less defined ordering* on pre/post-specifications. "Less defined" has a double meaning in this context: smaller domain-wise and vaguer range-wise. But such a linguistic consensus is not found in the underlying mathematics: $\vdash$ merges two opposite orderings, one pushing towards "smaller" specs and another to "larger" ones.

With the purpose to better understand this opposition, we decided to take advantage of a factorization of the refinement ordering which we found in [10, 14] but does not seem to have attracted much attention henceforth. Our approach to this result, which is calculational and *pointfree*-relational, contrasts with the hybrid models usually found in the literature, which typically use relational combinators to express definitions and properties but perform most reasoning steps at point-level, eg. using set-valued functions. A similar concern for pointfree relational reasoning can be found in [14], which we would like to study more in depth concerning the *demonic* calculus of relations.

The work reported in this paper should be regarded as a step towards a broader research aim: that of developing a clear-cut PF-theory of *coalgebraic* refinement. The intuition is provided by formula (1) once again, whose set-valued functions can be regarded *both* as power-transposes of binary relations [19] and coalgebras of the powerset functor. Instead of favouring the former view as in the current paper, we want to exploit the latter and follow the approach of [16], who study refinement of software components modelled by coalgebras of functor $\mathsf{F}X = (O \times (\mathsf{B}X))^I$, where $I$ and $O$ model inputs and outputs and $\mathsf{B}$ is a monad describing the component's behaviour pattern.

The approach has already been treated generically in the pointfree style [4], whereby set inclusion in (1) is generalized to a sub-preorder of $\mathsf{F}$-membership-based inclusion. There is, however, no coalgebraic counterpart to the $\vdash_{pre}$/$\vdash_{post}$ factorization studied in the current paper. Such a generalization is a prompt topic for future research.

## Acknowledgments

We thank Lindsay Groves for pointing us to reference [14] and the anonymous referees for helpful comments on the original submission. This research was carried out in the context of the PURE Project *(Program Understanding and Re-engineering: Calculi and Applications)* funded by FCT contract `POSI/ICHS/44304/2002`.

## References


1. C. Aarts, R. Backhouse, P. Hoogendijk, E. Voermans, and J. van der Woude. A relational theory of datatypes, Dec. 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. *Sci. of Comp. 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. L.S. Barbosa and J.N. Oliveira. Transposing partial components — an exercise on coalgebraic refinement. Technical report, DI/UM, Sep. 2005. (Submitted).
5. R. Bird and O. de Moor. *Algebra of Programming*. C.A.R. Hoare editor, Series in Computer Science. Prentice-Hall Int., 1997.
6. N. Boudriga, F. Elloumi, and A. Mili. On the lattice of specifications: Applications to a specification methodology. *Formal Asp. Comput.*, 4(6):544–571, 1992.
7. W.-P. de Roever, K. Engelhardt with the assistance of J. Coenen, K.-H. Buth, P. Gardiner, Y. Lakhnech, and F. Stomp. *Data Refinement Model-Oriented Proof methods and their Comparison*. Cambridge University Press, 1999. ISBN 0521641705.
8. M. Deutsch, M. Henson, and S. Reeves. Modular reasoning in Z: scrutinising monotonicity and refinement, 2006. Under consideration for publication in *Formal Asp. Comput.*.
9. M. Frappier. *A Relational Basis for Program Construction by Parts*. PhD thesis, University of Ottawa, 1995.
10. L. Groves. Refinement and the Z schema calculus. *ENTCS*, 70(3), 2002. Extended version available as Vict. Univ. of Wellington, CS Tech. Report CS-TR-02-31.
11. Jifeng He, C.A.R. Hoare, and J.W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *ESOP'86*, Springer LNCS (213), pages 187–196, 1986.
12. P. Hoogendijk. *A Generic Theory of Data Types*. PhD thesis, Univ. Eindhoven, NL, 1997.
13. C.B. Jones. *Software Development — A Rigorous Approach*. C.A.R. Hoare editor, Series in Computer Science. Prentice-Hall Int., 1980.
14. W. Kahl. Refinement and development of programs from relational specifications. *ENTCS*, 44(3):4.1–4.43, 2003.
15. E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, Inc., 6th ed., 1988.
16. Sun Meng and L.S. Barbosa. On refinement of generic state-based software components. In C. Rettray, S. Maharaj, and C. Shankland, editors, *10th Int. Conf. Algebraic Methods and Software Technology (AMAST'04)*, pages 506–520. Springer LNCS (3116). 2004.
17. C. Morgan. *Programming from Specification*. C.A.R. Hoare editor, Series in Computer Science. Prentice-Hall Int., 3rd edition, 1998.
18. S.C. Mu and R.S. Bird. Inverting functions as folds. In E. Boiten and B. Möller, editors, *6th Int. Conf. on Math. of Program Construction*, Springer LNCS (2386), pages 209–232. 2002.
19. J.N. Oliveira and C.J. Rodrigues. Transposing relations: from *Maybe* functions to hash tables. In *MPC'04 : 7th Int. Conf. on Math. of Program Construction*, Springer LNCS (3125), pages 334–356. 2004.
20. C.J. Rodrigues. Reificação e cálculos de reificação. Technical report, Universidade do Minho, April 1995. (In Portuguese).
21. C.J. Rodrigues. *Software Refinement by Calculation*. PhD thesis, Departamento de Informática, Universidade do Minho, 2006. (Forthcoming.).
22. J.M. Spivey. *The Z Notation — A Reference Manual*. C.A.R. Hoare editor, Series in Computer Science. Prentice-Hall Int., 1989.
23. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.