Reinvigorating pen-and-paper proofs in VDM: the pointfree approach

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

VDM'06 Workshop
Newcastle, 27-28 November 2006

Formal methods

Adopting a **formal** notation standard such as VDM-SL isn't enough:

- abstract models involve **conditions** which lead to
- **proof obligations** that need to be discharged

As in other branches of engineering

$$e = m + c$$

that is,

engineering = *model* first, then *calculate* ...

Calculate? Verify?

We know how to **calculate** since the school desk...

Formal methods

Adopting a **formal** notation standard such as VDM-SL isn't enough:

- abstract models involve **conditions** which lead to
- **proof obligations** that need to be discharged

As in other branches of engineering

$$e = m + c$$

that is,

engineering $=$ *<u>model</u> first, then <u>calculate</u>* ...

Calculate? Verify?

We know how to **calculate** since the school desk...

Formal methods

Adopting a **formal** notation standard such as VDM-SL isn't enough:

- abstract models involve **conditions** which lead to
- **proof obligations** that need to be discharged

As in other branches of engineering

$$e = m + c$$

that is,

engineering $=$ *<u>model</u> first, then <u>calculate</u>* ...

Calculate? Verify?

We know how to **calculate** since the school desk...

Tradition on “al-djabr” equational reasoning

Examples of “al-djabr” rules: in arithmetics

$$x - z \leq y \equiv x \leq y + z$$

In logics:

$$(x \wedge \neg z) \Rightarrow y \equiv x \Rightarrow (y \vee z)$$

“Al-djabr” rules are known since the 9c. (They are nowadays known as **Galois connections**.)

Question

Can VDM **proof obligations** be *calculated* along the same tradition?

Tradition on "al-djabr" equational reasoning

Examples of "al-djabr" rules: in arithmetics

$$x - z \leq y \equiv x \leq y + z$$

In logics:

$$(x \wedge \neg z) \Rightarrow y \equiv x \Rightarrow (y \vee z)$$

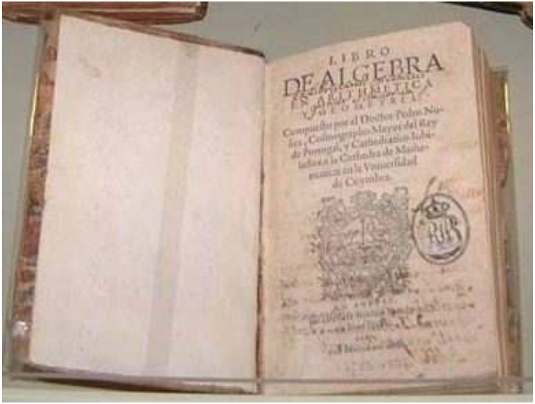
"Al-djabr" rules are known since the 9c. (They are nowadays known as **Galois connections**.)

Question

Can VDM **proof obligations** be *calculated* along the same tradition?

By the way

Nunes' *Libro de Algebra en Arithmetica y Geometria (1567)*



(...) ho inuẽtor desta arte foy hum Mathematico Mouro, cujo nome era Gebre, & ha em alguãs Liuarias hum pequeno tractado Arauigo, que contem os capitulos de q̃ usamos
(fol. a ij r)

Reference to *On the calculus of al-gabr and al-muqâbala* [1] by Abû Abd Allâh Muhamad B. Mûsâ Al-Huwârizmî, a famous 9c Persian mathematician.

[1] Original title: *Kitâb al-muhtasar fi hisab al-gabr wa-almuqâbala*.

Examples of proof obligations

The following are standard in VDM:

- **Satisfiability:** a *pre*/*post* pair is *satisfiable* iff

$$\forall\ a\ \cdot pre(a) \Rightarrow \exists\ b\ \cdot post(a, b) \tag{1}$$

- **Invariants:** in case the *pre*/*post* pair specifies an operation over a state with invariant **inv**,

$$\forall\ a\ \cdot pre(a) \Rightarrow \exists\ b\ \cdot inv(b) \wedge post(a, b) \tag{2}$$

  Moreover, invariants are to be maintained:

$$\forall\ b, a\ \cdot pre(a) \wedge post(a, b) \wedge inv(a) \Rightarrow inv(b) \tag{3}$$

Examples of proof obligations

The following are standard in VDM:

- **Satisfiability:** a *pre/post* pair is *satisfiable* iff

$$\forall\ a \ \cdot pre(a) \Rightarrow \exists\ b \ \cdot post(a, b) \tag{1}$$

- **Invariants:** in case the *pre*/*post* pair specifies an operation over a state with invariant **inv**,

$$\forall\ a \ \cdot pre(a) \Rightarrow \exists\ b \ \cdot inv(b) \wedge post(a, b) \tag{2}$$

  Moreover, invariants are to be maintained:

$$\forall\ b, a \ \cdot pre(a) \wedge post(a, b) \wedge inv(a) \Rightarrow inv(b) \tag{3}$$

Impact of (universal) quantification

Quantifiers:

- $\exists$ — easy to discharge (eg. by counter-examples)
- $\forall$ — hard to calculate with (in general), leading to (complex) inductive proofs.

What can we do about this?

- **Mechanical** proof support is one way
- Investigation of **alternative** calculation methods is another

An analogy:

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

How has traditional **engineering mathematics** tackled the complexity brought about by $\int$'s and $\partial/\partial x$'s?

Impact of (universal) quantification

Quantifiers:

- $\exists$ — easy to discharge (eg. by counter-examples)
- $\forall$ — hard to calculate with (in general), leading to (complex) inductive proofs.

What can we do about this?

- **Mechanical** proof support is one way
- Investigation of **alternative** calculation methods is another

An analogy:

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

How has traditional **engineering mathematics** tackled the complexity brought about by $\int$'s and $\partial/\partial x$'s?

The Laplace transform

$(\mathcal{L}\ f)s = \int_0^\infty e^{-st} f(t)dt$

$f(t)$	$\mathcal{L}(f)$
1	$\frac{1}{s}$
t	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^{n+1}}$
e^{at}	$\frac{1}{s-a}$
etc	

Pierre Laplace (1749-1827)

How it works

t-space	*s*-space
Given problem $y'' + 4y' + 3y = 0$ $y(0) = 3$ $y'(0) = 1$	**Subsidiary equation** $s^2 + 4sY + 3Y = 3s + 13$
Solution of given problem $y(t) = -2e^{-3t} + 5e^{-t}$	**Solution of subs. equation** $Y = \frac{-2}{s+3} + \frac{5}{s+1}$

An "s-space analog" for logical quantification

The pointfree ($\mathcal{PF}$) transform

ϕ	$\mathcal{PF}\ \phi$
$\langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle$	$b(R \cdot S)c$
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \forall\ x\ ::\ x\ R\ b \Rightarrow x\ S\ a \rangle$	$b(R \setminus S)a$
$\langle \forall\ c\ ::\ b\ R\ c \Rightarrow a\ S\ c \rangle$	$a(S\ /\ R)b$
$b\ R\ a \wedge c\ S\ a$	$(b, c)\langle R, S \rangle a$
$b\ R\ a \wedge d\ S\ c$	$(b, d)(R \times S)(a, c)$
$b\ R\ a \wedge b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ \ b)\ R\ (g\ \ a)$	$b(f^{\circ} \cdot R \cdot g)a$
$\textsc{True}$	$b\ \top\ a$
$\textsc{False}$	$b\ \bot\ a$

A transform for logic and set-theory

An old idea

$$\mathcal{PF}(\text{sets, predicates}) \quad = \quad \text{binary relations}$$

Calculus of binary relations

- 1860 - introduced by De Morgan, embryonic
- 1941 - Tarski's school, cf. *A Formalization of Set Theory without Variables*
- 1980's - coreflexive models of sets (Freyd and Scedrov, Eindhoven school)

Unifying approach

Everything is a (binary) relation

A transform for logic and set-theory

An old idea

$$\mathcal{PF}(\text{sets, predicates}) \quad = \quad \text{binary relations}$$

Calculus of binary relations

- 1860 - introduced by De Morgan, embryonic
- 1941 - Tarski's school, cf. *A Formalization of Set Theory without Variables*
- 1980's - coreflexive models of sets (Freyd and Scedrov, Eindhoven school)

Unifying approach

Everything is a (binary) relation

Binary Relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

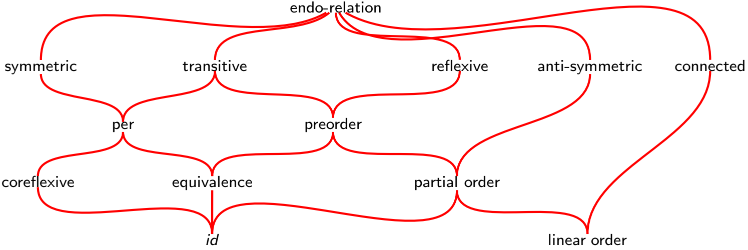
Converse of R — R° such that $a(R^\circ)b$ iff $b\ R\ a$.

Ordering

"$R \subseteq S$ — the "R is at most S" — the obvious $R \subseteq S$ **ordering**.

Binary relation taxonomy

Recall

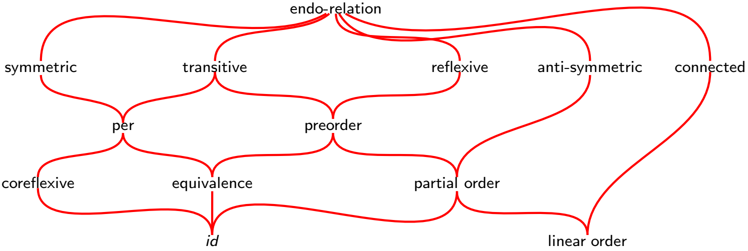


where a relation $A \xrightarrow{R} A$ is

reflexive:	iff $id_A \subseteq R$
coreflexive:	iff $R \subseteq id_A$
transitive:	iff $R \cdot R \subseteq R$
anti-symmetric:	iff $R \cap R^\circ \subseteq id_A$
symmetric:	iff $R \subseteq R^\circ (\equiv R = R^\circ)$
connected:	iff $R \cup R^\circ = \top$

Binary relation taxonomy

Recall

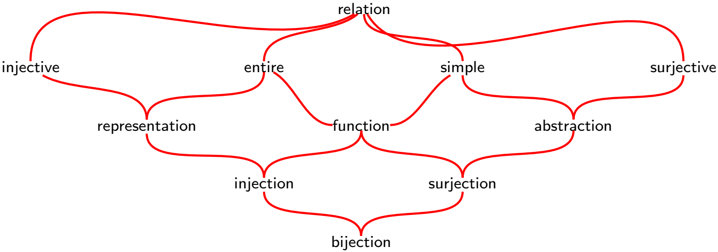


where a relation $A \xrightarrow{R} A$ is

reflexive:	iff $id_A \subseteq R$
coreflexive:	iff $R \subseteq id_A$
transitive:	iff $R \cdot R \subseteq R$
anti-symmetric:	iff $R \cap R^\circ \subseteq id_A$
symmetric:	iff $R \subseteq R^\circ (\equiv R = R^\circ)$
connected:	iff $R \cup R^\circ = \top$

Binary relation taxonomy

The whole picture:



where

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

$$\ker R = R^\circ \cdot R$$
$$\mathrm{img}\, R = R \cdot R^\circ$$

Functions in one slide

- A function f is a binary relation such that

Pointwise	Pointfree	
"Left" Uniqueness		
$b\ f\ a \wedge b'\ f\ a \quad \Rightarrow \quad b = b'$	$\mathrm{img}\, f \quad \subseteq \quad id$	(f is simple)
Leibniz principle		
$a = a' \quad \Rightarrow \quad f\ a = f\ a'$	$id \quad \subseteq \quad \ker f$	(f is entire)

- Back to useful "al-djabr" rules (GCs):

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

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

- Equality:

$$f \subseteq g \equiv f = g \equiv f \supseteq g$$

Functions in one slide

- A function f is a binary relation such that

Pointwise	Pointfree	
"Left" Uniqueness		
$b\ f\ a \wedge b'\ f\ a \quad \Rightarrow \quad b = b'$	$\mathrm{img}\ f \quad \subseteq \quad id$	(f is simple)
Leibniz principle		
$a = a' \quad \Rightarrow \quad f\ a = f\ a'$	$id \quad \subseteq \quad \ker f$	(f is entire)

- Back to useful "al-djabr" rules (GCs):

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

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

- Equality:

$$f \subseteq g \equiv f = g \equiv f \supseteq g$$

Simple relations in one slide

- "Al-djabr" rules for simple M:

$$M \cdot R \subseteq T \quad \equiv \quad (\delta\, M) \cdot R \quad \subseteq \quad M^\circ \cdot T \tag{4}$$

$$R \cdot M^\circ \subseteq T \quad \equiv \quad R \cdot \delta\, M \quad \subseteq \quad T \cdot M \tag{5}$$

where

$$\delta\, R \quad = \quad \ker R \cap id$$

(=domain of R) is the coreflexive part of $\ker R$.

- **Equality**

$$M = N \equiv \quad M \subseteq N \wedge \delta\, N \subseteq \delta\, M \tag{6}$$

follows from (4, 5).

Simple relations in one slide

- "Al-djabr" rules for simple M:

$$M \cdot R \subseteq T \quad \equiv \quad (\delta\, M) \cdot R \quad \subseteq \quad M^\circ \cdot T \tag{4}$$

$$R \cdot M^\circ \subseteq T \quad \equiv \quad R \cdot \delta\, M \quad \subseteq \quad T \cdot M \tag{5}$$

where

$$\delta\, R \quad = \quad \ker R \cap id$$

(=domain of R) is the coreflexive part of $\ker R$.

- **Equality**

$$M = N \equiv \quad M \subseteq N \wedge \delta\, N \subseteq \delta\, M \tag{6}$$

follows from (4, 5).

Predicates PF-transformed

- **Binary** predicates :

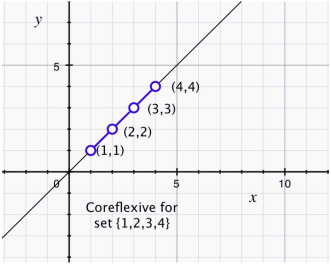
$$R = [\![b]\!] \quad \equiv \quad (y \ R \ x \equiv b(y,x))$$

- **Unary** predicates become fragments of *id* (coreflexives) :

$$R = [\![p]\!] \quad \equiv \quad (y \ R \ x \equiv (p \ x) \land x = y)$$

eg.

$$[\![1 \leq x \leq 4]\!] =$$



Boolean algebra of coreflexives

$$
\begin{aligned}
[\![p \wedge q]\!] &= [\![p]\!] \cdot [\![q]\!] && (7)\\
[\![p \vee q]\!] &= [\![p]\!] \cup [\![q]\!] && (8)\\
[\![\neg p]\!] &= id - [\![p]\!] && (9)\\
[\![false]\!] &= \bot && (10)\\
[\![true]\!] &= id && (11)
\end{aligned}
$$

Note the very useful fact that **conjunction** of coreflexives is **composition**

LPF versus PF-transform

Example

PF-calculation of “partial” implication [5]:

$$\forall\ i, j \in \mathbb{Z}\ \cdot i \geq j \Rightarrow subp(i, j) = i - j$$

where

$$\begin{aligned} &subp : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \\ &subp(i, j) \triangleq \ \ if\ \ i = j\ \ then\ 0\ \ else\ 1 + subp(i, j + 1) \end{aligned}$$

Simplicity "does it all" — I think

First step — calculate its PF-transform:

$$
\begin{array}{ll}
& (i,j) \in \delta\, Subp \Rightarrow (i-j)\ Subp\ (i,j) \\
\equiv & \quad \{\ \text{PF-transform rule}\quad (f\ b)\ R\ (g\ a) \ \equiv\ b(f^\circ \cdot R \cdot g)a \quad \} \\
& \delta\, Subp \ \subseteq\ (-)^\circ \cdot Subp \\
\equiv & \quad \{\ \text{converses}\ \} \\
& \delta\, Subp \ \subseteq\ Subp^\circ \cdot (-) \\
\equiv & \quad \{\ \text{“al-djabr” (simple relations)}\ \} \\
& Subp \ \subseteq\ (-)
\end{array}
$$

Second step: calculate $Subp \subseteq (-)$, see overleaf

Does $Subp \subseteq (-)$ hold?

We draw

$$
\begin{array}{l}
subp : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \\
subp(i,j) \triangleq \;\; if \;\; i = j \;\; then \;\; 0 \;\; else \;\; 1 + subp(i, j+1)
\end{array}
$$

in a “divide & conqueur” diagram:

$$
\begin{tikzcd}
\mathbb{Z} \times \mathbb{Z} \arrow[r, "D"] \arrow[d, "Subp"'] & 1 + \mathbb{Z} \times \mathbb{Z} \arrow[d, "id+Subp"] \\
\mathbb{Z} & 1 + \mathbb{Z} \arrow[l, "c"]
\end{tikzcd}
$$

where
$$
\begin{array}{lcl}
\Delta & = & \lambda\, x.(x, x) \\
D & = & [\Delta \cdot !^{\circ} \,, id \times (-1)]^{\circ} \\
c & = & [\underline{0} \,, (1+)]
\end{array}
$$

Thus

$$
Subp = \mu X.(c \cdot (id + X) \cdot D))
$$

Does $Subp \subseteq (-)$ hold?

We draw

$$
\begin{aligned}
& subp : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \\
& subp(i,j) \triangleq \ \ \mathit{if}\ \ i = j\ \ \mathit{then}\ 0\ \ \mathit{else}\ 1 + subp(i, j+1)
\end{aligned}
$$

in a "divide & conqueur" diagram:

$$
\begin{tikzcd}
\mathbb{Z} \times \mathbb{Z} \arrow[r, "D"] \arrow[d, "Subp"'] & 1 + \mathbb{Z} \times \mathbb{Z} \arrow[d, "id+Subp"] \\
\mathbb{Z} & 1 + \mathbb{Z} \arrow[l, "c"]
\end{tikzcd}
\quad \text{where} \quad
\begin{array}{rcl}
\Delta & = & \lambda\, x.(x,x) \\
D & = & [\Delta \cdot !^{\circ}\ , id \times (-1)]^{\circ} \\
c & = & [\underline{0}\ , (1+)]
\end{array}
$$

Thus

$$Subp \quad = \quad \mu X.(c \cdot (id + X) \cdot D))$$

Does $Subp \subseteq (-)$ hold?

Our calculation is based on the **fixpoint rule**:

$$\mu g \subseteq X \quad \Leftarrow \quad g\ X \subseteq X \tag{12}$$

as follows

$$
\begin{array}{ll}
& Subp \subseteq (-) \\
\Leftarrow & \quad \{ \text{ fixpoint rule , for } g\ X = c \cdot (id + X) \cdot D \ \} \\
& c \cdot (id + (-)) \cdot D \subseteq (-) \\
\equiv & \quad \{ \text{ unfold } c \text{ and } D \ \} \\
& [\underline{0}\ , (1+) \cdot (-)] \cdot [\Delta \cdot !^{\circ}\ , id \times (-1)]^{\circ} \subseteq (-) \\
\equiv & \quad \{ \text{ converses and coproducts } \ \}
\end{array}
$$

Calculate implication

$$
\begin{array}{ll}
& \underline{0} \cdot \Delta^\circ \;\cup\; (1+) \cdot (-) \cdot (id \times (-1))^\circ \quad \subseteq \quad (-) \\
\equiv & \qquad \{ \text{ ``al-djabr''s of } \cup \text{ and functions } \} \\
& \begin{array}{rcl} \underline{0} & = & (-) \cdot \Delta \\ (1+) \cdot (-) & = & (-) \cdot (id \times (-1)) \end{array} \\
\equiv & \qquad \{ \text{ go pointwise } \} \\
& \begin{array}{rcl} 0 & = & i - i \\ 1 + (i - j) & = & i - (j - 1) \end{array} \\
\equiv & \qquad \{ \text{ arithmetics } \} \\
& \mathit{true}
\end{array}
$$

In fact, it can be further shown that the implication is an equivalence — let us see how:

The other side of the equivalence

$$\forall\ i,j \in \mathbb{Z}\ \cdot subp(i,j) = i - j \Rightarrow i \geq j$$

$\equiv$ { PF-transform }

$$(-)^\circ \cdot Subp \cap id \quad \subseteq \quad \delta\, Subp$$

$\Leftarrow$ { Dedekind ; domain is the coreflexive part of kernel }

$$((-)^\circ \cap Subp^\circ) \cdot Subp \quad \subseteq \quad Subp^\circ \cdot Subp$$

$\equiv$ { converses ; $Subp \subseteq (-)$, as calculated above }

$$Subp^\circ \cdot Subp \quad \subseteq \quad Subp^\circ \cdot Subp$$

$\equiv$ { trivial }

$$true$$

Proof obligations (PF-transformed)

Let

$$\begin{array}{rcll} Inv & = & [\![inv]\!] & \text{(a coreflexive)} \\ Pre & = & [\![pre]\!] & \text{(a coreflexive)} \\ Post & = & [\![post]\!] & \end{array}$$

in

$$Spec \triangleq Post \cdot Pre$$

and recall eg.

$$\forall\ a\ \cdot pre(a) \Rightarrow \exists\ b\ \cdot post(a, b) \tag{13}$$

$$\forall\ b, a\ \cdot pre(a) \wedge post(a, b) \wedge inv(a) \Rightarrow inv(b) \tag{14}$$

Then

Proof obligations (PF-transformed)

Let

$$
\begin{array}{rcll}
Inv & = & [\![inv]\!] & \text{(a coreflexive)} \\
Pre & = & [\![pre]\!] & \text{(a coreflexive)} \\
Post & = & [\![post]\!] &
\end{array}
$$

in

$$Spec \triangleq Post \cdot Pre$$

and recall eg.

$$\forall\ a\ \cdot pre(a) \Rightarrow \exists\ b\ \cdot post(a, b) \qquad (13)$$

$$\forall\ b, a\ \cdot pre(a) \wedge post(a, b) \wedge inv(a) \Rightarrow inv(b) \qquad (14)$$

Then

Proof obligations (PF-transformed)

1. **Satisfiability** — (13) PF-transforms to

$$Pre \quad \subseteq \quad \delta\, Post \tag{15}$$

equivalent to

$$Pre \quad \subseteq \quad \top \cdot Post$$

2. **Invariants** — (14) PF-transforms to

$$\rho\,(Spec \cdot Inv) \subseteq Inv \tag{16}$$

equivalent to

$$Spec \cdot Inv \quad \subseteq \quad Inv \cdot Spec \tag{17}$$

Proof obligations (PF-transformed)

1. **Satisfiability** — (13) PF-transforms to

$$Pre \quad \subseteq \quad \delta\, Post \tag{15}$$

equivalent to

$$Pre \quad \subseteq \quad \top \cdot Post$$

2. **Invariants** — (14) PF-transforms to

$$\rho\,(Spec \cdot Inv) \subseteq Inv \tag{16}$$

equivalent to

$$Spec \cdot Inv \quad \subseteq \quad Inv \cdot Spec \tag{17}$$

Proof obligations (PF-transformed)

Functions

The special case of (17) where *Spec* is a function f,

$$f \cdot Inv \quad \subseteq \quad Inv \cdot f \tag{18}$$

maps back to the pointwise

$$\forall\ a \ \cdot inv(a) \Rightarrow inv(f(a)) \tag{19}$$

Invariants in general

In general, let $A \xrightarrow{Spec} B$ be a spec over two datatypes A and B each with its invariant, say Φ and Ψ, respectively. Then (18) generalizes to

$$Spec \cdot \Phi \;\subseteq\; \Psi \cdot Spec \tag{20}$$

We will write

$$\Phi \xrightarrow{Spec} \Psi \tag{21}$$

to mean $Spec \cdot \Phi \;\subseteq\; \Psi \cdot Spec$. Thus,

1. invariants can be regarded as **types** and
2. invariant preservation can be re-written as a **type discipline**, eg.

$$\frac{\Phi \xrightarrow{R} \Psi \quad , \quad \Psi \xrightarrow{S} \Gamma}{\Phi \xrightarrow{S \cdot R} \Gamma} \tag{22}$$

(composition),

Invariants in general

In general, let $A \xrightarrow{Spec} B$ be a spec over two datatypes A and B each with its invariant, say Φ and Ψ, respectively. Then (18) generalizes to

$$Spec \cdot \Phi \ \subseteq \ \Psi \cdot Spec \tag{20}$$

We will write

$$\Phi \xrightarrow{Spec} \Psi \tag{21}$$

to mean $Spec \cdot \Phi \ \subseteq \ \Psi \cdot Spec$. Thus,

1. invariants can be regarded as **types** and
2. invariant preservation can be re-written as a **type discipline**, eg.

$$\frac{\Phi \xrightarrow{R} \Psi \quad , \quad \Psi \xrightarrow{S} \Gamma}{\Phi \xrightarrow{S \cdot R} \Gamma} \tag{22}$$

(composition),

Invariants in general

In general, let $A \xrightarrow{Spec} B$ be a spec over two datatypes A and B each with its invariant, say Φ and Ψ, respectively. Then (18) generalizes to

$$Spec \cdot \Phi \;\subseteq\; \Psi \cdot Spec \tag{20}$$

We will write

$$\Phi \xrightarrow{Spec} \Psi \tag{21}$$

to mean $Spec \cdot \Phi \;\subseteq\; \Psi \cdot Spec$. Thus,

1. invariants can be regarded as **types** and
2. invariant preservation can be re-written as a **type discipline**, eg.

$$\frac{\Phi \xrightarrow{R} \Psi \quad , \quad \Psi \xrightarrow{S} \Gamma}{\Phi \xrightarrow{S \cdot R} \Gamma} \tag{22}$$

(composition),

Invariants “are” types

$$\frac{\Phi \xrightarrow{R} \Psi \ , \Phi' \subseteq \Phi}{\Phi' \xrightarrow{R} \Psi} \quad , \quad \frac{\Psi' \subseteq \Psi, \ \Phi \xrightarrow{R} \Psi'}{\Phi \xrightarrow{R} \Psi} \tag{23}$$

(sub-typing), etc

Compare this **invariants-as-types** PF-theory with

Quoting [4], p.116

The valid objects of Datec are those which (...) satisfy inv-Datec. This has a profound consequence for the type mechanism of the notation. (...) The inclusion of a sub-typing mechanism which allows truth-valued functions forces the type checking here to rely on proofs.

Invariants “are” types

$$\frac{\Phi \xrightarrow{R} \Psi \;,\; \Phi' \subseteq \Phi}{\Phi' \xrightarrow{R} \Psi} \quad , \quad \frac{\Psi' \subseteq \Psi,\; \Phi \xrightarrow{R} \Psi'}{\Phi \xrightarrow{R} \Psi} \tag{23}$$

(sub-typing), etc

Compare this **invariants-as-types** PF-theory with

Quoting [4], p.116

The valid objects of Datec are those which (...) satisfy inv-Datec. This has a profound consequence for the type mechanism of the notation. (...) The inclusion of a sub-typing mechanism which allows truth-valued functions forces the type checking here to rely on proofs.

Data structures PF-transformed

- Relational databases resort to the mathematical notion of a **relation** to model **data**.

  Why not do the same in VDM?

- In the sequel we regard VDM finite mappings ($A \xrightarrow{\sim} B$) as **simple** relations and resort to "al-djabr" rules to prove invariant preservation

- Why?
  - No need for **induction**
  - Proofs don't even require **finiteness**
  - (Quite a few) results of the standard VDM theory of mappings
    - **extend** further to arbitrary binary relations
    - are **equivalences**, not just implications

Data structures PF-transformed

- Relational databases resort to the mathematical notion of a **relation** to model **data**.

  Why not do the same in VDM?

- In the sequel we regard VDM finite mappings ($A \tilde{\rightarrow} B$) as **simple** relations and resort to "al-djabr" rules to prove invariant preservation

- Why?
  - No need for **induction**
  - Proofs don't even require **finiteness**
  - (Quite a few) results of the standard VDM theory of mappings
    - **extend** further to arbitrary binary relations
    - are **equivalences**, not just implications

VDM mappings are finite **simple** relations

This leads to a PF-transformed mapping theory, eg.

Mapping comprehension

$$\{g(a) \mapsto f(M(a)) \mid a \in dom\ M\}$$

PF-transforms to

$$f \cdot M \cdot g^{\circ} \tag{24}$$

However

Need to ensure simplicity of the comprehension, see next slide

VDM mappings are finite **simple** relations

This leads to a PF-transformed mapping theory, eg.

Mapping comprehension

$$\{g(a) \mapsto f(M(a)) \mid a \in dom\ M\}$$

PF-transforms to

$$f \cdot M \cdot g^{\circ} \tag{24}$$

However

Need to ensure simplicity of the comprehension, see next slide

Mapping comprehension — "simple" simplicity argument

$$
\begin{array}{ll}
& f \cdot M \cdot g^{\circ} \cdot (f \cdot M \cdot g^{\circ})^{\circ} \subseteq id \\
\equiv & \qquad \{ \text{ converses } \} \\
& f \cdot M \cdot g^{\circ} \cdot g \cdot M^{\circ} \cdot f^{\circ} \subseteq id \\
\equiv & \qquad \{ \text{ “al-djabr” } \} \\
& M \cdot g^{\circ} \cdot g \cdot M^{\circ} \subseteq f^{\circ} \cdot f \\
\equiv & \qquad \{ \text{ definition of kernel of a relation } \} \\
& \ker (g \cdot M^{\circ}) \subseteq \ker f \\
\equiv & \qquad \{ \text{ injectivity preorder } R \leq S \equiv \ker S \subseteq \ker R \ \} \\
& f \leq g \cdot M^{\circ}
\end{array}
$$

That is to say, M satisfies the $g \rightarrow f$ *functional dependency* [6] (always fine wherever g is injective).

Straight from the VDM-SL on-line manual

Operator	Name	Semantics description
`m1 † m2`	Override	overrides and merges `m1` with `m2`, i.e. it is like a merge except that `m1` and `m2` need not be compatible; any common elements are as by `m2` (so `m2` overrides `m1`.)

PF (formal) **semantics**:

$$[\![m_1 \dagger m_2]\!] = [\![m_2]\!] \rightarrow [\![m_2]\!] \, , \, [\![m_1]\!]$$

which resorts to the relational version of **McCarthy** conditional:

$$R \rightarrow S \, , \, T \stackrel{\mathrm{def}}{=} (S \cdot \delta R) \cup (T \cdot \neg \delta R)$$

Straight from the VDM-SL on-line manual

Operator	Name	Semantics description
`m1 † m2`	Override	overrides and merges `m1` with `m2`, i.e. it is like a merge except that `m1` and `m2` need not be compatible; any common elements are as by `m2` (so `m2` overrides `m1`.)

PF (formal) **semantics**:

$$[\![m_1 \dagger m_2]\!] \quad = \quad [\![m_2]\!] \rightarrow [\![m_2]\!] \ , \ [\![m_1]\!]$$

which resorts to the relational version of **McCarthy** conditional:

$$R \rightarrow S \ , \ T \quad \stackrel{\mathrm{def}}{=} \quad (S \cdot \delta\, R) \cup (T \cdot \neg \delta\, R)$$

Mapping override

From PF-definition

$$M \dagger N \stackrel{\text{def}}{=} N \rightarrow N \;,\; M \tag{25}$$

equivalent to

$$M \dagger N = N \cup M \cdot (\neg \delta\, N) \tag{26}$$

it is easy to show

$$M \dagger M = M \tag{27}$$

$$M \dagger \perp = \perp \dagger M = M \tag{28}$$

More generally, **equivalences**

$$N \subseteq M \equiv M \dagger N = M \tag{29}$$

$$\delta\, M \subseteq \delta\, N \equiv M \dagger N = N \tag{30}$$

hold.

Override is associative (Lemma 6.7 in [4] — †-ass)

$$
\begin{array}{cl}
 & (R \dagger S) \dagger P \\
= & \quad \{ \ (25) \text{ twice } \} \\
 & P \to P \ , \ (S \to S \ , \ R) \\
= & \quad \{ \ (26) \text{ twice } \} \\
 & P \cup (S \cup R \cdot (\neg\delta\, S)) \cdot (\neg\delta\, P) \\
= & \quad \{ \ \text{distribution ; de Rorgan } \} \\
 & P \cup S \cdot (\neg\delta\, P) \cup R \cdot (\neg(\delta\, S \cup \delta\, P)) \\
= & \quad \{ \ (26) \text{ ; domain of override } \} \\
 & (S \dagger P) \cup R \cdot (\neg\delta\, (S \dagger P)) \\
= & \quad \{ \ (26) \ \} \\
 & R \dagger (S \dagger P)
\end{array}
$$

Important

- Holds for **arbitrary** relations
- **No** need of **induction**

Override is associative (Lemma 6.7 in [4] — †-ass)

$$
\begin{array}{cl}
& (R \dagger S) \dagger P \\
= & \qquad \{ \ (25)\ \text{twice} \ \} \\
& P \rightarrow P \ , \ (S \rightarrow S \ , \ R) \\
= & \qquad \{ \ (26)\ \text{twice} \ \} \\
& P \cup (S \cup R \cdot (\neg \delta\, S)) \cdot (\neg \delta\, P) \\
= & \qquad \{ \ \text{distribution ; de Rorgan} \ \} \\
& P \cup S \cdot (\neg \delta\, P) \cup R \cdot (\neg (\delta\, S \cup \delta\, P)) \\
= & \qquad \{ \ (26)\ ;\ \text{domain of override} \ \} \\
& (S \dagger P) \cup R \cdot (\neg \delta\, (S \dagger P)) \\
= & \qquad \{ \ (26) \ \} \\
& R \dagger (S \dagger P)
\end{array}
$$

Important

- Holds for **arbitrary** relations
- **No** need of **induction**

The ubiquitous finite mapping

Usual **"design paterns"** in VDM modelling:

- **Classification:** $A \tilde{\rightharpoonup} B$ where the type of interest is A and B is a classifier

  Cf. recording (partial) equivalence relations [4]: $\ker M = R^\circ \cdot R$ for M simple is always a per (partial equivalence relation).

- **Quantification:** $Bag\ A \triangleq A \tilde{\rightharpoonup} \mathbb{N}$ (bags, orders, invoices etc)
- **Identification:** $K \tilde{\rightharpoonup} A$ where A is the TOI and K is a space of **keys** (eg. name-spaces, database entities, objects, etc)
- **Heaps**: $K \tilde{\rightharpoonup} \mathsf{F}(A, K)$ where K is an address space (eg. in modelling memory management)

PF-transformed invariants

Typical *invariant patterns* associated to the *identification* design pattern are

- **Referential integrity:**

$$M \quad \preceq \quad N \qquad \text{or} \qquad M^\circ \quad \preceq \quad N$$

  where $\preceq$ denotes the **mapping definition** partial order

$$M \preceq N \quad = \quad \delta\, M \subseteq \delta\, N \tag{31}$$

- **Range-wise property:** because the TOI is in the range, a typical VDM invariant pattern arises, $\forall\ a \in rng\ M \ \cdot \psi(a)$ which PF-transforms to

$$M \subseteq \Psi \cdot M \tag{32}$$

CRUD = identification + persistence

CRUD?

Wikipedia

In computing, **CRUD** *is an acronym for Create, Read, Update, and Delete. (...) It is used as a shorthand way to refer to the four basic functions of* **persistence,** *which is a major part of nearly all computer software.*

CRUD on mapping M:

- $Create(N)$: $M \mapsto N \dagger M$
- $Read(a)$: b such that $b\ M\ a$
- $Update(f, \Phi)$: $M \mapsto M \dagger f \cdot M \cdot \Phi$
- $Delete(\Phi)$: $M \mapsto M \cdot (\neg\Phi)$

Example of proof discharge by PF-calculation: **range-wise** invariant preservation by (selective) **update**

CRUD = identification + persistence

CRUD?

Wikipedia

In computing, **CRUD** *is an acronym for Create, Read, Update, and Delete. (...) It is used as a shorthand way to refer to the four basic functions of* **persistence,** *which is a major part of nearly all computer software.*

CRUD on mapping M:

- *Create*(N): $M \mapsto N \dagger M$
- *Read*(a): b *such that* $b\ M\ a$
- *Update*(f, Φ): $M \mapsto M \dagger f \cdot M \cdot \Phi$
- *Delete*(Φ): $M \mapsto M \cdot (\neg\Phi)$

Example of proof discharge by PF-calculation: **range-wise** invariant preservation by (selective) **update**

Selective update

Notation shorthand

$$M^f_\Phi \quad \triangleq \quad M \dagger f \cdot M \cdot \Phi \tag{33}$$

Very easy to show:

$$M^{id}_\Phi = M \tag{34}$$
$$M^f_\perp = M \tag{35}$$
$$M^f_{id} = f \cdot M \tag{36}$$

Now, how does selective update ($\_^f_\Phi$) preserve

$$inv\ M \quad \triangleq \quad M \subseteq \Psi \cdot M$$

Proof discharge by PF-calculation

We have to find conditions for $(\_^f_\Phi)$ to bear type

$$Inv \xrightarrow{(\_^f_\Phi)} Inv \tag{37}$$

Since $(\_^f_\Phi\ )$ is a function, the proof discharge is easy (19), for all M:

$$
\begin{array}{cl}
& inv(M) \quad \Rightarrow \quad inv(M^f_\Phi)) \\
\equiv & \qquad \{ \text{ expand } inv(M) \ \} \\
& M \subseteq \Psi \cdot M \quad \Rightarrow \quad M^f_\Phi \subseteq \Psi \cdot M^f_\Phi \\
\equiv & \qquad \{ \text{ since } \Psi \cdot M \subseteq M \ \} \\
& M = \Psi \cdot M \quad \Rightarrow \quad M^f_\Phi \subseteq \Psi \cdot M^f_\Phi
\end{array}
$$

So we focus on $M^f_\Phi \subseteq \Psi \cdot M^f_\Phi$, assuming $M = \Psi \cdot M$:

Proof discharge by PF-calculation

$$M^f_\Phi \subseteq \Psi \cdot M^f_\Phi$$

$\equiv$ { (33) twice }

$$M \dagger f \cdot M \cdot \Phi \subseteq \Psi \cdot (M \dagger f \cdot M \cdot \Phi)$$

$\equiv$ { $M = \Psi \cdot M$; distribution }

$$(\Psi \cdot M) \dagger f \cdot (\Psi \cdot M) \cdot \Phi \subseteq (\Psi \cdot M) \dagger (\Psi \cdot f \cdot M \cdot \Phi)$$

$\Leftarrow$ { monotonicity }

$$f \cdot \Psi \subseteq \Psi \cdot f$$

$\equiv$ { (21) — of course! }

$$\Psi \xrightarrow{f} \Psi$$

Other variations on mappings

Mapping aliasing

In computing, *aliasing* means multiple names for the same data location.

VDM (pointwise)

$$alias(a, b, M) \triangleq$$
$$M \dagger (\ if\ \ b \in dom\ M\ \ then\ \{a \mapsto M(b)\}\ else\ \{\mapsto\}\ )$$

PF-transform

$$alias(a, b, M) \triangleq\ M \dagger M \cdot \underline{b} \cdot \underline{a}^{\circ}$$

where $\underline{a}$ and $\underline{b}$ are constant functions.

Other variations on mappings

Mapping aliasing

In computing, *aliasing* means multiple names for the same data location.

VDM (pointwise)

$$
\begin{aligned}
&\mathit{alias}(a, b, M) \triangleq \\
&\quad M \dagger (\ \mathit{if}\ \ b \in \mathit{dom}\ M\ \ \mathit{then}\ \{a \mapsto M(b)\}\ \mathit{else}\ \{\mapsto\}\ )
\end{aligned}
$$

PF-transform

$$
\mathit{alias}(a, b, M) \triangleq\ M \dagger M \cdot \underline{b} \cdot \underline{a}^{\circ}
$$

where $\underline{a}$ and $\underline{b}$ are constant functions.

Aliasing

Notation shorthand

$M_{a:=b}$ for $M \dagger M \cdot \underline{b} \cdot \underline{a}^\circ$ (suggestive of eg. regarding M as a piece of memory and a and b variable names or addresses.)

Sample properties

- **Identity**:

$$M_{a:=a} = M \tag{38}$$

- **Idempotency**:

$$(M_{a:=b})_{a:=b} = M_{a:=b} \tag{39}$$

both instances of

$$M_{a:=b} = M \equiv M \cdot \underline{b} \subseteq M \cdot \underline{a} \tag{40}$$

Aliasing

Notation shorthand

$M_{a:=b}$ for $M \dagger M \cdot \underline{b} \cdot \underline{a}^\circ$ (suggestive of eg. regarding M as a piece of memory and a and b variable names or addresses.)

Sample properties

- **Identity**:

$$M_{a:=a} \quad = \quad M \tag{38}$$

- **Idempotency**:

$$(M_{a:=b})_{a:=b} \quad = \quad M_{a:=b} \tag{39}$$

both instances of

$$M_{a:=b} = M \quad \equiv \quad M \cdot \underline{b} \subseteq M \cdot \underline{a} \tag{40}$$

Aliasing

Notation shorthand

$M_{a:=b}$ for $M \dagger M \cdot \underline{b} \cdot \underline{a}^{\circ}$ (suggestive of eg. regarding M as a piece of memory and a and b variable names or addresses.)

Sample properties

- **Identity**:

$$M_{a:=a} \quad = \quad M \tag{38}$$

- **Idempotency**:

$$(M_{a:=b})_{a:=b} \quad = \quad M_{a:=b} \tag{39}$$

both instances of

$$M_{a:=b} = M \quad \equiv \quad M \cdot \underline{b} \subseteq M \cdot \underline{a} \tag{40}$$

Equating extends aliasing

Let us move on to the **classification** design pattern, and recall the problem of *Recording equivalence relations* [4]:

Equate a and b

VDM:

$$
\begin{array}{l}
equate(a, b, M) \triangleq \\
\quad M \dagger \{x \mapsto M(b)) \mid x \in \ dom\ M \wedge M(x) = M(a)\}
\end{array}
$$

PF-transform

$$equate(a, b, M) \triangleq \ M \dagger M \cdot \underline{b} \cdot \underline{a}^\circ \cdot (\ker M)$$

Thus *equate* is an "evolution" of *aliasing*, equivalent to

$$M \dagger (M \cdot \underline{b}) \cdot (M \cdot \underline{a})^\circ \cdot M$$

Equating extends aliasing

Let us move on to the **classification** design pattern, and recall the problem of *Recording equivalence relations* [4]:

Equate a and b

VDM:

$$
\begin{array}{l}
equate(a, b, M) \triangleq \\
\quad M \dagger \{x \mapsto M(b)) \mid x \in \ dom\ M \wedge M(x) = M(a)\}
\end{array}
$$

PF-transform

$$
equate(a, b, M) \triangleq \ M \dagger M \cdot \underline{b} \cdot \underline{a}^{\circ} \cdot (\ker M)
$$

Thus *equate* is an "evolution" of *aliasing*, equivalent to

$$
M \dagger (M \cdot \underline{b}) \cdot (M \cdot \underline{a})^{\circ} \cdot M
$$

Equating extends aliasing

Let us move on to the **classification** design pattern, and recall the problem of *Recording equivalence relations* [4]:

Equate a and b

VDM:

$$\begin{array}{l} equate(a, b, M) \triangleq \\ \quad M \dagger \{x \mapsto M(b)) \mid x \in \ dom\ M \wedge M(x) = M(a)\} \end{array}$$

PF-transform

$$equate(a, b, M) \triangleq \ M \dagger M \cdot \underline{b} \cdot \underline{a}^{\circ} \cdot (\ker M)$$

Thus *equate* is an "evolution" of *aliasing*, equivalent to

$$M \dagger (M \cdot \underline{b}) \cdot (M \cdot \underline{a})^{\circ} \cdot M$$

Reasoning about *equate*

Abstraction function

Two mappings M, N represent the same PER iff

$$\ker M \quad = \quad \ker N$$

(ker is the abstraction function)

Properties of *equate*

Writing $M_{a \simeq b}$ as abbreviation of $M \dagger (M \cdot \underline{b}) \cdot (M \cdot \underline{a})^{\circ} \cdot M$:

$$M_{a \simeq a} \quad = \quad M \tag{41}$$

$$\ker M_{a \simeq b} \quad = \quad \ker M_{b \simeq a} \tag{42}$$

and so on.

Reasoning about *equate*

Abstraction function

Two mappings M, N represent the same PER iff

$$\ker M \quad = \quad \ker N$$

($\ker$ is the abstraction function)

Properties of *equate*

Writing $M_{a \simeq b}$ as abbreviation of $M \dagger (M \cdot \underline{b}) \cdot (M \cdot \underline{a})^\circ \cdot M$:

$$M_{a \simeq a} \quad = \quad M \tag{41}$$

$$\ker M_{a \simeq b} \quad = \quad \ker M_{b \simeq a} \tag{42}$$

and so on.

Summary

- Learn with the other engineering disciplines
- Rôle of PF-patterns (advantage of "writing less symbols"), eg. easier to spot *al-djabr* rule
- Shift from "implication first" to "calculational" logic

  "Chase" equivalence : bad use of implication-first logic may lead to "50% loss in theory"

- PF-transform: need for a cultural "shift"?

Inspiration

- John Backus *Algebra of Programs* (1978) [2]
- Binary relations already in Cliff's thesis (1981) [3]
- Bird-Meertens-Backhouse approach [1]

Context

- **Coalgebraic** semantics for **components** and objects
- Possibly applicable to VDM(++)
- **Invariants** regarded as coreflexive **bisimulations** in the underlying coalgebra theory
- Finite mappings PF-reasoning relates to on-going work in **database** theory "refactoring" [6]

Current work

- Impact of partial predicates in PF-transform (LPP instead of LPF?)
- Foundations: which approach to undefinedness? LPF [5]? Dijkstra/Scholten's (and variations thereof)? [7]
- Prospect for tool support:
  - RelView (Kiel)
  - 'G'ALCULATOR project (Minho)

Limitations of *RELVIEW*

- *RELVIEW* only works on relations with finite domains.
- Relations between elements have to be explicitly defined.
- Thus, it is very specific and not usable in the general cases.
- We need a more generic tool ...

Galculator

- *Galculator* implements relation algebra.
- Relational calculus is done by expression manipulation.
- Manipulation is performed by a strategic typed term-rewriting system implemented using **Haskell** and GADTs.
- Galois connections are used as rewriting rules allowing the exploitation of proofs by indirect equality.

Closing

"Algebra (...) is thing causing admiration"

> *(...) "Mainly because we see often a great Mathematician unable to resolve a question by Geometrical means, and solve it by Algebra, being that same Algebra taken from Geometry, which is thing causing admiration."*

— my (literal, not literary) translation of:

> *(...) Principalmente que vemos algumas vezes, no poder vn gran Mathematico resoluer vna question por medios Geometricos, y resolverla por Algebra, siendo la misma Algebra sacada de la Geometria, q̃ es cosa de admiraciõ.*

[**Pedro Nunes** (1502-1578) in **Libro de Algebra en Arithmetica y Geometria**, 1567, fols. 270–270v.]

Closing

"Algebra (...) is thing causing admiration"

> *(...) "Mainly because we see often a great Mathematician unable to resolve a question by Geometrical means, and solve it by Algebra, being that same Algebra taken from Geometry, which is thing causing admiration."*

— my (literal, not literary) translation of:

> *(...) Principalmente que vemos algumas vezes, no poder vn gran Mathematico resoluer vna question por medios Geometricos, y resolverla por Algebra, siendo la misma Algebra sacada de la Geometria, q̃ es cosa de admiraciõ.*

[**Pedro Nunes** (1502-1578) in **Libro de Algebra en Arithmetica y Geometria**, 1567, fols. 270–270v.]

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

J. Backus.
Can programming be liberated from the von Neumann style? a functional style and its algebra of programs.
, 21(8):613–639, August 1978.

C.B. Jones.
Development Methods for Computer Programs including a Notion of Interference.
PhD thesis, Oxford University, June 1981.
Printed as: Programming Research Group, Technical Monograph 25.

C.B. Jones.
Systematic Software Development Using VDM.
Series in Computer Science. Prentice-Hall International, 1986.

C.A. R. Hoare.

C.B. Jones.
Reasoning about partial functions in the formal development of programs.
pages 3–25. ENTCS, volume 145, Elsevier, 2006.

J.N. Oliveira.
Pointfree foundations for lossless decomposition, 2006.
Draft of paper in preparation.

B. Schieder and M. Broy.
Adapting calculational logic to the undedfined.
The Computer Journal, 42(2):74–81, 1999.