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 = 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 = model first, then calculate* . . .

**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:

\[ (x \land \neg z) \Rightarrow y \equiv x \Rightarrow (y \lor z) \]

“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:

\[ (x \land \neg z) \Rightarrow y \equiv x \Rightarrow (y \lor z) \]

“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)

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

\[ \forall a \cdot pre(a) \Rightarrow \exists b \cdot post(a, b) \]  

(1)

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

\[ \forall a \cdot pre(a) \Rightarrow \exists b \cdot inv(b) \land post(a, b) \]  

(2)

Moreover, invariants are to be maintained:

\[ \forall b, a \cdot pre(a) \land post(a, b) \land inv(a) \Rightarrow inv(b) \]  

(3)
Examples of proof obligations

The following are standard in VDM:

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

  \[
  \forall a \cdot pre(a) \Rightarrow \exists b \cdot post(a, b) \quad (1)
  \]

- **Invariants**: in case the \( pre/post \) pair specifies an operation over a state with invariant \( \text{inv} \),

  \[
  \forall a \cdot pre(a) \Rightarrow \exists b \cdot inv(b) \land post(a, b) \quad (2)
  \]

  Moreover, invariants are to be maintained:

  \[
  \forall b, a \cdot pre(a) \land post(a, b) \land inv(a) \Rightarrow inv(b) \quad (3)
  \]
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:

- ∃ — easy to discharge (eg. by counter-examples)
- ∀ — 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

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

<table>
<thead>
<tr>
<th>( f(t) )</th>
<th>( \mathcal{L}(f) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{1}{s} )</td>
</tr>
<tr>
<td>t</td>
<td>( \frac{1}{s^2} )</td>
</tr>
<tr>
<td>( t^n )</td>
<td>( \frac{n!}{s^{n+1}} )</td>
</tr>
<tr>
<td>( e^{at} )</td>
<td>( \frac{1}{s-a} )</td>
</tr>
</tbody>
</table>

Pierre Laplace (1749-1827)
How it works

t-space

Given problem

\[ y'' + 4y' + 3y = 0 \]
\[ y(0) = 3 \]
\[ y'(0) = 1 \]

Subsidiary equation

\[ s^2 Y + 4sY + 3Y = 3s + 13 \]

Solution of given problem

\[ y(t) = -2e^{-3t} + 5e^{-t} \]

s-space

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

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\mathcal{PF} \phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \exists a : : b \ R \ a \land a \ S \ c \rangle$</td>
<td>$b(R \cdot S)c$</td>
</tr>
<tr>
<td>$\langle \forall a, b : : b \ R \ a \Rightarrow b \ S \ a \rangle$</td>
<td>$R \subseteq S$</td>
</tr>
<tr>
<td>$\langle \forall a : : a \ R \ a \rangle$</td>
<td>$id \subseteq R$</td>
</tr>
<tr>
<td>$\langle \forall x :: x \ R \ b \Rightarrow x \ S \ a \rangle$</td>
<td>$b(R \setminus S)a$</td>
</tr>
<tr>
<td>$\langle \forall c :: b \ R \ c \Rightarrow a \ S \ c \rangle$</td>
<td>$a(S / R)b$</td>
</tr>
<tr>
<td>$b \ R \ a \land c \ S \ a$</td>
<td>$(b, c)\langle R, S\rangle a$</td>
</tr>
<tr>
<td>$b \ R \ a \land d \ S \ c$</td>
<td>$(b, d)(R \times S)(a, c)$</td>
</tr>
<tr>
<td>$b \ R \ a \land b \ S \ a$</td>
<td>$b \ (R \cap S) \ a$</td>
</tr>
<tr>
<td>$b \ R \ a \lor b \ S \ a$</td>
<td>$b \ (R \cup S) \ a$</td>
</tr>
<tr>
<td>$(f \ b) \ R \ (g \ a)$</td>
<td>$b(f \circ \cdot R \cdot g)a$</td>
</tr>
<tr>
<td>$\text{TRUE}$</td>
<td>$b \top a$</td>
</tr>
<tr>
<td>$\text{FALSE}$</td>
<td>$b \bot a$</td>
</tr>
</tbody>
</table>
A transform for logic and set-theory

An old idea

\[ \mathcal{PF}(\text{sets, predicates}) = \text{binary relations} \]

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

\[ \mathcal{PF}(\text{sets, predicates}) = \text{binary relations} \]

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 $R$ — $R^\circ$ 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 $R \cup R^\circ = \top$
Where a relation \( A \xrightarrow{R} A \) is:

- **reflexive:** \( \text{iff } \text{id}_A \subseteq R \)
- **coreflexive:** \( \text{iff } R \subseteq \text{id}_A \)
- **transitive:** \( \text{iff } R \cdot R \subseteq R \)
- **anti-symmetric:** \( \text{iff } R \cap R^\circ \subseteq \text{id}_A \)
- **symmetric:** \( \text{iff } R \subseteq R^\circ (\equiv R = R^\circ) \)
- **connected:** \( \text{iff } R \cup R^\circ = \top \)
Binary relation taxonomy

The whole picture:

where

<table>
<thead>
<tr>
<th></th>
<th>Reflexive</th>
<th>Coreflexive</th>
</tr>
</thead>
<tbody>
<tr>
<td>ker $R$</td>
<td>entire $R$</td>
<td>injective $R$</td>
</tr>
<tr>
<td>img $R$</td>
<td>surjective $R$</td>
<td>simple $R$</td>
</tr>
</tbody>
</table>

$\ker R = R^\circ \cdot R$

$\text{img} R = R \cdot R^\circ$
Functions in one slide

- A function $f$ is a binary relation such that

<table>
<thead>
<tr>
<th>Pointwise</th>
<th>Pointfree</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Left&quot; Uniqueness</td>
<td>$(f$ is simple$)$</td>
</tr>
<tr>
<td>$b f a \land b' f a \Rightarrow b = b'$</td>
<td>$\text{img } f \subseteq id$</td>
</tr>
<tr>
<td>Leibniz principle</td>
<td>$(f$ is entire$)$</td>
</tr>
<tr>
<td>$a = a' \Rightarrow f a = f a'$</td>
<td>$id \subseteq \ker f$</td>
</tr>
</tbody>
</table>

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

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S$$
$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f$$

- Equality:

$$f \subseteq g \equiv f = g \equiv f \supseteq g$$
Functions in one slide

• A function $f$ is a binary relation such that

<table>
<thead>
<tr>
<th>Pointwise</th>
<th>Pointfree</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Left” Uniqueness</td>
<td>(f is simple)</td>
</tr>
<tr>
<td>$b f a \land b' f a \Rightarrow b = b'$</td>
<td>$\text{img } f \subseteq \text{id}$</td>
</tr>
<tr>
<td>Leibniz principle</td>
<td>(f is entire)</td>
</tr>
<tr>
<td>$a = a' \Rightarrow f a = f a'$</td>
<td>$\text{id} \subseteq \ker f$</td>
</tr>
</tbody>
</table>

• Back to useful “al-djabr” rules (GCs):

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S$$

$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f$$

• Equality:

$$f \subseteq g \equiv f = g \equiv f \supseteq g$$
Simple relations in one slide

- “Al-djabr” rules for simple $M$:

$$M \cdot R \subseteq T \equiv (\delta M) \cdot R \subseteq M^\circ \cdot T \quad (4)$$

$$R \cdot M^\circ \subseteq T \equiv R \cdot \delta M \subseteq T \cdot M \quad (5)$$

where

$$\delta R \equiv \ker R \cap id$$

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

- Equality

$$M = N \equiv M \subseteq N \land \delta N \subseteq \delta M \quad (6)$$

follows from (4, 5).
**Simple relations** in one slide

- “Al-djabr” rules for simple $M$:

  \[ M \cdot R \subseteq T \equiv (\delta M) \cdot R \subseteq M^\circ \cdot T \quad (4) \]

  \[ R \cdot M^\circ \subseteq T \equiv R \cdot \delta M \subseteq T \cdot M \quad (5) \]

  where

  \[ \delta R = \ker R \cap id \]

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

- **Equality**

  \[ M = N \equiv M \subseteq N \land \delta N \subseteq \delta M \quad (6) \]

  follows from (4, 5).
Predicates PF-transformed

- **Binary** predicates:
  \[ R = \left[ b \right] \equiv (y \ R \ x \equiv b(y, x)) \]

- **Unary** predicates become fragments of \( id \) (coreflexives):
  \[ R = \left[ p \right] \equiv (y \ R \ x \equiv (p \ x) \land x = y) \]

eg.

\[ \left[ 1 \leq x \leq 4 \right] = \]
Boolean algebra of coreflexives

\[ [p \land q] = [p] \cdot [q] \]  \hspace{1cm} (7)
\[ [p \lor q] = [p] \cup [q] \]  \hspace{1cm} (8)
\[ [\neg p] = id - [p] \]  \hspace{1cm} (9)
\[ [false] = \bot \]  \hspace{1cm} (10)
\[ [true] = id \]  \hspace{1cm} (11)

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

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

\[ \forall i, j \in \mathbb{Z} \cdot i \geq j \Rightarrow subp(i, j) = i - j \] (12)

where

\[ subp : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \]

\[ subp(i, j) \triangleq \text{if } i = j \text{ then } 0 \text{ else } 1 + subp(i, j + 1) \]
Simplicity “does it all” — I think

First step — calculate its PF-transform:

\[ i \geq j \Rightarrow (i - j) \text{Subp}(i, j) \]
\[ \equiv \{ \text{PF-transform rule } (f \circ b) \ R \ (g \ a) \equiv b(f^\circ \cdot R \cdot g)a \} \]
\[ \delta \text{Subp} \subseteq \ (-) \circ \cdot \text{Subp} \]
\[ \equiv \{ \text{converses} \} \]
\[ \delta \text{Subp} \subseteq \text{Subp}^\circ \cdot (-) \]
\[ \equiv \{ \text{“al-djabr” (simple relations)} \} \]
\[ \text{Subp} \subseteq (-) \]

Second step: calculate \( \text{Subp} \subseteq (-) \), see overleaf
Does $Subp \subseteq (\_\_)$ hold?

We draw

$$subp : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$subp(i, j) \triangleq \text{if } i = j \text{ then } 0 \text{ else } 1 + subp(i, j + 1)$$

in a “divide & conquer” diagram:

\[
\begin{array}{ccc}
\mathbb{Z} \times \mathbb{Z} & \xrightarrow{D} & 1 + \mathbb{Z} \times \mathbb{Z} \\
\downarrow Subp & & \downarrow id + Subp \\
\mathbb{Z} & \xleftarrow{c} & 1 + \mathbb{Z}
\end{array}
\]

Thus

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

where

\[
\begin{align*}
\Delta &= \lambda x.(x, x) \\
D &= [\Delta \cdot !^\circ, id \times (-1)]^\circ \\
c &= [0, (1+)]
\end{align*}
\]
Does $Subp \subseteq (\cdot)$ hold?

We draw

$$subp : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$subp(i, j) \triangleq \text{if } i = j \text{ then } 0 \text{ else } 1 + subp(i, j + 1)$$

in a “divide & conquer” diagram:

Thus

$$Subp = \mu X.(c \cdot (id + X) \cdot D)$$
Does $Subp \subseteq (−)$ hold?

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

\[ \mu g \subseteq X \iff g \cdot X \subseteq X \]  \hspace{1cm} (13)

as follows

\[
\begin{align*}
Subp \quad & \subseteq \quad (−) \\
\iff & \quad \{ \text{fixpoint rule, for } g \cdot X = c \cdot (id + X) \cdot D \} \\
& \quad \{ c \cdot (id + (−)) \cdot D \subseteq (−) \\
& \quad \equiv \quad \{ \text{unfold } c \text{ and } D \} \\
& \quad \{ [0, (1+) \cdot (−)] \cdot [\Delta \cdot !^\circ, id \times (−1)]^\circ \subseteq (−) \\
& \quad \equiv \quad \{ \text{converses and coproducts} \}
\end{align*}
\]
Calculate implication

\[ 0 \cdot \Delta^\circ \cup (1+) \cdot (\mathbf{1}) \cdot (id \times (-1))^\circ \subseteq (-) \]
\[ \equiv \{ \, \text{“al-djabr”s of } \cup \text{ and functions} \, \} \]
\[ \begin{align*}
0 &= (-) \cdot \Delta \\
(1+) \cdot (\mathbf{1}) &= (-) \cdot (id \times (-1))
\end{align*} \]
\[ \equiv \{ \, \text{go pointwise} \, \} \]
\[ \begin{align*}
0 &= i - i \\
1 + (i - j) &= i - (j - 1)
\end{align*} \]
\[ \equiv \{ \, \text{arithmetics} \, \} \]
\[ true \]

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

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

\[ \equiv \{ \text{PF-transform} \} \]

\[ (\neg)^\circ \cdot \text{Subp} \cap \text{id} \subseteq \delta \text{Subp} \]

\[ \Leftarrow \{ \text{Dedekind ; domain is the coreflexive part of kernel} \} \]

\[ ((\neg)^\circ \cap \text{Subp}^\circ) \cdot \text{Subp} \subseteq \text{Subp}^\circ \cdot \text{Subp} \]

\[ \equiv \{ \text{converses ; Subp} \subseteq (\neg), \text{as calculated above} \} \]

\[ \text{Subp}^\circ \cdot \text{Subp} \subseteq \text{Subp}^\circ \cdot \text{Subp} \]

\[ \equiv \{ \text{trivial} \} \]

\[ \text{true} \]
Proof obligations (PF-transformed)

Let

\[
\begin{align*}
\text{Inv} & = \llbracket \text{inv} \rrbracket \\
\text{Pre} & = \llbracket \text{pre} \rrbracket \\
\text{Post} & = \llbracket \text{post} \rrbracket
\end{align*}
\]

in

\[\text{Spec} \triangledown \text{Post} \cdot \text{Pre}\]

and recall eg.

\[
\forall a \cdot \text{pre}(a) \Rightarrow \exists b \cdot \text{post}(a, b) \quad (14)
\]

\[
\forall b, a \cdot \text{pre}(a) \land \text{post}(a, b) \land \text{inv}(a) \Rightarrow \text{inv}(b) \quad (15)
\]

Then
Proof obligations (PF-transformed)

Let

\[ \text{Inv} = \{ \text{inv} \} \quad (\text{a coreflexive}) \]
\[ \text{Pre} = \{ \text{pre} \} \quad (\text{a coreflexive}) \]
\[ \text{Post} = \{ \text{post} \} \]

in

\[ \text{Spec} \triangleq \text{Post} \cdot \text{Pre} \]

and recall eg.

\[ \forall a \cdot \text{pre}(a) \Rightarrow \exists b \cdot \text{post}(a, b) \quad (14) \]
\[ \forall b, a \cdot \text{pre}(a) \land \text{post}(a, b) \land \text{inv}(a) \Rightarrow \text{inv}(b) \quad (15) \]

Then
Proof obligations (PF-transformed)

1. **Satisfiability** — (14) PF-transforms to

   \[ \text{Pre} \subseteq \delta \text{Post} \quad (16) \]

   equivalent to

   \[ \text{Pre} \subseteq \top \cdot \text{Post} \]

2. **Invariants** — (15) PF-transforms to

   \[ \rho (\text{Spec} \cdot \text{Inv}) \subseteq \text{Inv} \quad (17) \]

   equivalent to

   \[ \text{Spec} \cdot \text{Inv} \subseteq \text{Inv} \cdot \text{Spec} \quad (18) \]
Proof obligations (PF-transformed)

1. **Satisfiability** — (14) PF-transforms to

\[ Pre \subseteq \delta \ Post \]  \hspace{2cm} (16)

equivalent to

\[ Pre \subseteq \top \cdot Post \]

2. **Invariants** — (15) PF-transforms to

\[ \rho (Spec \cdot Inv) \subseteq Inv \]  \hspace{2cm} (17)

equivalent to

\[ Spec \cdot Inv \subseteq Inv \cdot Spec \]  \hspace{2cm} (18)
Proof obligations (PF-transformed)

Functions
The special case of (18) where Spec is a function $f$,

$$f \cdot \text{Inv} \subseteq \text{Inv} \cdot f$$

maps back to the pointwise

$$\forall a \cdot \text{inv}(a) \Rightarrow \text{inv}(f(a))$$
Invariants in general

In general, let \( A \xrightarrow{Spec} B \) be a spec over two datatypes \( A \) and \( B \) each with its invariant, say \( \Phi \) and \( \Psi \), respectively. Then (19) generalizes to

\[
Spec \cdot \Phi \subseteq \Psi \cdot Spec
\]  

(21)

We will write

\[
\Phi \xrightarrow{Spec} \Psi
\]  

(22)

to mean (21). Thus,

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

\[
\Phi \xrightarrow{R} \Psi , \quad \Psi \xrightarrow{S} \Gamma \quad \Rightarrow \quad \Phi \xrightarrow{S \cdot R} \Gamma
\]  

(23)

(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 $\Phi$ and $\Psi$, respectively. Then (19) generalizes to

$$Spec \cdot \Phi \subseteq \Psi \cdot Spec$$  \hspace{1cm} (21)

We will write

$$\Phi \xrightarrow{Spec} \Psi$$  \hspace{1cm} (22)

to mean (21). Thus,

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

$$\Phi \xrightarrow{R} \Psi, \quad \Psi \xrightarrow{S} \Gamma$$

$$\Phi \xrightarrow{S \cdot R} \Gamma$$  \hspace{1cm} (23)

(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 \( \Phi \) and \( \Psi \), respectively. Then (19) generalizes to

\[
Spec \cdot \Phi \subseteq \Psi \cdot Spec
\]  
(21)

We will write

\[
\Phi \xrightarrow{Spec} \Psi
\]
(22)

to mean (21). Thus,

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

\[
\Phi \xrightarrow{R} \Psi , \quad \Psi \xrightarrow{S} \Gamma
\]
\[
\Phi \xrightarrow{S \cdot R} \Gamma
\]
(23)

(composition),
Invariants “are” types

\[
\begin{align*}
\Phi \xrightarrow{R} \Psi, \Phi' \subseteq \Phi \\
\Phi' \xrightarrow{R} \Psi' \\
\Psi' \subseteq \Psi, \Phi \xrightarrow{R} \Psi' \\
\Phi \xrightarrow{R} \Psi
\end{align*}
\]

(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

\[
\begin{align*}
\Phi \xrightarrow{R} \Psi , \Phi' \subseteq \Phi \\
\Phi' \xrightarrow{R} \Psi \\
\psi' \subseteq \psi , \phi \xrightarrow{R} \psi' \\
\phi \xrightarrow{R} \psi
\end{align*}
\]

(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 \textit{relation} to model \textbf{data}.
  
  \textit{Why not do the same in VDM?}

- In the sequel we regard VDM finite mappings \((A \leadsto B)\) as \textit{simple} relations and resort to “al-djabr” rules to prove invariant preservation

- Why?
  - No need for \textit{induction}
  - Proofs don’t even require \textit{finiteness}
  - (Quite a few) results of the standard VDM theory of mappings
    - extend further to arbitrary binary relations
    - are \textit{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 $(A \sim B)$ 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
VDM mappings are finite \textbf{simple} relations

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

**Mapping comprehension**

\[
\{ g(a) \mapsto f(M(a)) \mid a \in \text{dom } M \}
\]

PF-transforms to

\[
f \cdot M \cdot g^\circ \tag{25}
\]

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**

\[ \{ g(a) \mapsto f(M(a)) \mid a \in \text{dom } M \} \]

PF-transforms to

\[ f \cdot M \cdot g^\circ \]  \hspace{1cm} (25)

**However**

Need to ensure simplicity of the comprehension, see next slide
Mapping comprehension — “simple” simplicity argument

\[ f \cdot M \cdot g^\circ \cdot (f \cdot M \cdot g^\circ)^\circ \subseteq id \]

\[ \equiv \quad \{ \text{converses} \} \]

\[ f \cdot M \cdot g^\circ \cdot g \cdot M^\circ \cdot f^\circ \subseteq id \]

\[ \equiv \quad \{ \text{“al-djabr”} \} \]

\[ M \cdot g^\circ \cdot g \cdot M^\circ \subseteq f^\circ \cdot f \]

\[ \equiv \quad \{ \text{definition of kernel of a relation} \} \]

\[ \ker (g \cdot M^\circ) \subseteq \ker f \]

\[ \equiv \quad \{ \text{injectivity preorder} \ R \leq S \equiv \ker S \subseteq \ker R \} \]

\[ f \leq g \cdot M^\circ \]

That is to say, \( M \) satisfies the \( g \rightarrow f \) \textit{functional dependency} \cite{6} (always fine wherever \( g \) is injective).
Straight from the VDM-SL on-line manual

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Semantics description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m₁ † m₂</td>
<td>Override</td>
<td>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.)</td>
</tr>
</tbody>
</table>

PF (formal) semantics:

\[
\left[ [m_1 \uplus m_2] \right] = \left[ [m_2] \rightarrow [m_2] \right] \cup [m_1]
\]

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

\[
R \rightarrow S \ , \ T \ \overset{\text{def}}{=} (S \cdot \delta R) \cup (T \cdot \neg \delta R)
\]
Straight from the VDM-SL on-line manual

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Semantics description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1 \uparrow m_2$</td>
<td>Override</td>
<td>overrides and merges $m_1$ with $m_2$, i.e. it is like a merge except that $m_1$ and $m_2$ need not be compatible; any common elements are as by $m_2$ (so $m_2$ overrides $m_1$.)</td>
</tr>
</tbody>
</table>

PF (formal) **semantics**: 

$$[[m_1 \uparrow m_2]] = [[[m_2]] \rightarrow [[[m_2]]], [[[m_1]]]]$$

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

$$R \rightarrow S , \ T \overset{\text{def}}{=} (S \cdot \delta R) \cup (T \cdot \neg \delta R)$$
Mapping override

From PF-definition

\[ M \dagger N \triangleq N \rightarrow N , M \] (26)

equivalent to

\[ M \dagger N = N \cup M \cdot (\neg \delta N) \] (27)

it is easy to show

\[ M \dagger M = M \] (28)

\[ M \dagger \bot = \bot \dagger M = M \] (29)

More generally, the following equivalences hold:

\[ N \subseteq M \equiv M \dagger N = M \] (30)

\[ \delta M \subseteq \delta N \equiv M \dagger N = N \] (31)
Override is associative (Lemma 6.7 in [4] — $\vdash$-ass)

\[(R \vdash S) \vdash P\]

\[= \quad \{ \text{(26) twice} \} \]

\[P \rightarrow P, \ (S \rightarrow S, \ R)\]

\[= \quad \{ \text{(27) twice} \} \]

\[P \cup (S \cup R \cdot (\neg \delta S)) \cdot (\neg \delta P)\]

\[= \quad \{ \text{distribution ; de Morgan} \} \]

\[P \cup S \cdot (\neg \delta P) \cup R \cdot (\neg (\delta S \cup \delta P))\]

\[= \quad \{ \text{(27) ; domain of override} \} \]

\[(S \vdash P) \cup R \cdot (\neg \delta (S \vdash P))\]

\[= \quad \{ \text{(27)} \} \]

\[R \vdash (S \vdash P)\]
Override is associative (Lemma 6.7 in [4] — †-ass)

\[(R \uparrow S) \uparrow P\]

= \{ (26) twice \}

\[P \to P, (S \to S, R)\]

= \{ (27) twice \}

\[P \cup (S \cup R \cdot (\neg \delta S)) \cdot (\neg \delta P)\]

= \{ distribution ; de Morgan \}

\[P \cup S \cdot (\neg \delta P) \cup R \cdot (\neg (\delta S \cup \delta P))\]

= \{ (27) ; domain of override \}

\[(S \uparrow P) \cup R \cdot (\neg \delta (S \uparrow P))\]

= \{ (27) \}

\[R \uparrow (S \uparrow P)\]

Important

- Holds for arbitrary relations
- No need of induction
The ubiquitous finite mapping

Usual “design patterns” in VDM modelling:

- **Classification:** $A \leadsto B$ 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:** $\text{Bag } A \triangle A \leadsto N$ (bags, orders, invoices etc)

- **Identification:** $K \leadsto A$ where $A$ is the TOI and $K$ is a space of keys (eg. name-spaces, database entities, objects, etc)

- **Heaps:** $K \leadsto F(A, K)$ 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:**

  \[ M \preceq N \quad \text{or} \quad M^\circ \preceq N \]

  where \( \preceq \) denotes the *mapping definition* partial order

  \[ M \preceq N = \delta M \subseteq \delta N \quad (32) \]

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

  \[ M \subseteq \Psi \cdot M \quad (33) \]
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**($N$): $M \mapsto N \uparrow M$
- **Read**($a$): $b$ such that $b \ M \ a$
- **Update**($f, \Phi$): $M \mapsto M \uparrow f \cdot M \cdot \Phi$
- **Delete**($\Phi$): $M \mapsto M \cdot (\neg \Phi)$

Example of proof discharge by PF-calculation: range-wise invariant preservation by (selective) update
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($N$):** $M \mapsto N \uparrow M$
- **Read($a$):** $b$ such that $b \in M \land a$
- **Update($f, \Phi$):** $M \mapsto M \uparrow f \cdot M \cdot \Phi$
- **Delete($\Phi$):** $M \mapsto M \cdot (\neg \Phi)$

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

Notation shorthand

\[ M^f_\Phi \triangleq M \upharpoonright f \cdot M \cdot \Phi \] (34)

Very easy to show:

\[ M^i_d \Phi = M \] (35)
\[ M^f_\perp = M \] (36)
\[ M^f_\text{id} = f \cdot M \] (37)

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

\[ \text{inv } M \triangleq M \subseteq \Psi \cdot M \]
Proof discharge by PF-calculation

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

\[
\text{Inv} \xrightarrow{(\_\Phi^f)} \text{Inv}
\]

(38)

Since \( (\_\Phi^f) \) is a function, the proof discharge is easy (20), for all \( M \):

\[
\text{inv}(M) \Rightarrow \text{inv}(M_{\Phi}^f))
\]

\[
\equiv \left\{ \text{expand } \text{inv}(M) \right\}
\]

\[
M \subseteq \Psi \cdot M \Rightarrow M_{\Phi}^f \subseteq \Psi \cdot M_{\Phi}^f
\]

\[
\equiv \left\{ \text{since } \Psi \cdot M \subseteq M \right\}
\]

\[
M = \Psi \cdot M \Rightarrow M_{\Phi}^f \subseteq \Psi \cdot M_{\Phi}^f
\]

So we focus on \( M_{\Phi}^f \subseteq \Psi \cdot M_{\Phi}^f \), assuming \( M = \Psi \cdot M \):
Proof discharge by PF-calculation

\[ M^f_\Phi \subseteq \psi \cdot M^f_\Phi \]

\[ \equiv \quad \{ \text{twice (34)} \} \]

\[ M \vdash f \cdot M \cdot \Phi \subseteq \psi \cdot (M \vdash f \cdot M \cdot \Phi) \]

\[ \equiv \quad \{ \text{\(M = \psi \cdot M\); distribution (*)} \} \]

\[ (\psi \cdot M) \vdash f \cdot (\psi \cdot M) \cdot \Phi \subseteq (\psi \cdot M) \vdash (\psi \cdot f \cdot M \cdot \Phi) \]

\[ \Leftarrow \quad \{ \text{monotonicity} \} \]

\[ f \cdot \psi \subseteq \psi \cdot f \]

\[ \equiv \quad \{ \text{(22) — of course!} \} \]

\[ \psi \xrightarrow{f} \psi \]
Comments

Step (*) above relies on

\[ \Phi \cdot (R \dagger S) = (\Phi \cdot R) \dagger (\Phi \cdot S) \iff S \preceq \Phi \cdot S \]

(39)

whose proof is

\[ \Phi \cdot (R \dagger S) \]

\[ = \quad \{ \text{McCarthy conditional} \} \]

\[ S \rightarrow \Phi \cdot S , \Phi \cdot R \]

\[ = \quad \{ \delta (\delta S) = \delta S \} \]

\[ \delta S \rightarrow \Phi \cdot S , \Phi \cdot R \]

\[ = \quad \{ \text{side-condition} \} \]

\[ \delta (\Phi \cdot S) \rightarrow \Phi \cdot S , \Phi \cdot R \]

\[ = \quad \{ \} \]

\[ (\Phi \cdot R) \dagger (\Phi \cdot S) \]
Thus, we still have to discharge

$$f \cdot M \cdot \Phi \preceq \Psi \cdot f \cdot M \cdot \Phi$$  \hspace{1cm} (40)

Equivalent to

$$M \cdot \Phi \preceq \delta (\Psi \cdot f) \cdot M \cdot \Phi$$

This is left as exercise to the reader.
Other variations on mappings

Mapping aliasing

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

VDM (pointwise)

\[
\text{alias}(a, b, M) \equiv \\
M \upharpoonright ( \begin{array}{l}
\text{if } b \in \text{dom } M \text{ then } \{a \mapsto M(b)\} \\
\text{else } \{\mapsto\}
\end{array})
\]

PF-transform

\[
\text{alias}(a, b, M) \equiv M \upharpoonright M \cdot b \cdot a^\circ
\]

where \(a\) and \(b\) are constant functions.
Other variations on mappings

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

VDM (pointwise)

\[
\text{alias}(a, b, M) \triangleq M \upharpoonright ( \quad \text{if } b \in \text{dom } M \text{ then } \{ a \mapsto M(b) \} \quad \text{else } \{ \mapsto \} \quad )
\]

PF-transform

\[
\text{alias}(a, b, M) \triangleq M \upharpoonright M \cdot b \cdot a^\circ
\]

where \(a\) and \(b\) are constant functions.
Aliasing

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

Sample properties

- **Identity:**

  $$M_{a:=a} = M$$ (41)

- **Idempotency:**

  $$\left(M_{a:=b}\right)_{a:=b} = M_{a:=b}$$ (42)

both instances of

$$M_{a:=b} = M \equiv M \cdot b \subseteq M \cdot a$$ (43)
Aliasing

Notation shorthand

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

Sample properties

- **Identity:**

\[
M_{a:=a} = M
\]  
(41)

- **Idempotency:**

\[
(M_{a:=b})_{a:=b} = M_{a:=b}
\]  
(42)

both instances of

\[
M_{a:=b} = M \equiv M \cdot b \subseteq M \cdot a
\]  
(43)
Aliasing

Notation shorthand
\( M_{a:=b} \) for \( M \uparrow M \cdot b \cdot a^o \) (suggestive of eg. regarding \( M \) as a piece of memory and \( a \) and \( b \) variable names or addresses.)

Sample properties

- **Identity:**

  \[
  M_{a:=a} = M \quad (41)
  \]

- **Idempotency:**

  \[
  (M_{a:=b})_{a:=b} = M_{a:=b} \quad (42)
  \]

both instances of

\[
M_{a:=b} = M \equiv M \cdot b \subseteq M \cdot a \quad (43)
\]
Comments

Calculation of (43):

\[ M_{a:=b} = M \]

\[ \equiv \{ \text{expanding shorthand} \} \]

\[ M \upharpoonright M \cdot b \cdot a^\circ = M \]

\[ \equiv \{ (30) \} \]

\[ M \cdot b \cdot a^\circ \subseteq M \]

\[ \equiv \{ \text{“al-djabr”} \} \]

\[ M \cdot b \subseteq M \cdot a \]

(41) follows immediately from (43). (42) is not so immediate but also easy to calculate.

\[ (M_{a:=b})_{a:=b} = M_{a:=b} \]

\[ \equiv \{ (43) \} \]

\[ (M_{a:=b}) \cdot b \subseteq (M_{a:=b}) \cdot a \]

etc
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:

\[
equate(a, b, M) \triangleq M \uparrow \{ x \mapsto M(b) \mid x \in \text{dom } M \land M(x) = M(a) \}
\]

PF-transform

\[
equate(a, b, M) \triangleq M \uparrow M \cdot b \cdot a^\circ \cdot (\ker M)
\]

Thus equate is an “evolution” of aliasing, equivalent to

\[
M \uparrow (M \cdot b) \cdot (M \cdot a)^\circ \cdot M
\]
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:**

\[
equate(a, b, M) \triangleq M \upharpoonright \{ x \mapsto M(b) \mid x \in \text{dom } M \land M(x) = M(a) \}
\]

**PF-transform**

\[
equate(a, b, M) \triangleq M \upharpoonright M \cdot b \cdot a^\circ \cdot (\ker M)
\]

Thus `equate` is an “evolution” of aliasing, equivalent to

\[
M \upharpoonright (M \cdot b) \cdot (M \cdot a)^\circ \cdot M
\]
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:

\[
equate(a, b, M) \triangleq \\
M \upharpoonright \{x \mapsto M(b) \mid x \in \text{dom } M \land M(x) = M(a)\}
\]

**PF-transform**

\[
equate(a, b, M) \triangleq M \upharpoonright M \cdot b \cdot a^\circ \cdot (\ker M)
\]

Thus *equate* is an “evolution” of *aliasing*, equivalent to

\[
M \upharpoonright (M \cdot b) \cdot (M \cdot a)^\circ \cdot M
\]
Reasoning about *equate*

**Abstraction function**

Two mappings $M, N$ represent the same PER iff

$$\ker M = \ker N$$

($\ker$ is the abstraction function)

**Properties of *equate***

Writing $M_{a\sim b}$ as abbreviation of $M \uparrow (M \cdot b) \cdot (M \cdot a) \circ M$:

$$M_{a\sim a} = M$$  \hspace{1cm} (44)

$$\ker M_{a\sim b} = \ker M_{b\sim a}$$ \hspace{1cm} (45)

and so on.
Reasoning about _equate_

**Abstraction function**
Two mappings $M, N$ represent the same PER iff

$$\ker M = \ker N$$

(ker is the abstraction function)

**Properties of _equate_**
Writing $M_{a\sim b}$ as abbreviation of $M \uparrow (M \cdot b) \cdot (M \cdot a)^\circ \cdot M$:

$$M_{a\sim a} = M$$  (44)

$$\ker M_{a\sim b} = \ker M_{b\sim a}$$  (45)

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]
- 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 resolver vna question por medios Geometricos, y resolverla por Algebra, siendo la misma Algebra sacada de la Geometria, ñ es cosa de admiraciõ.

[ Pedro Nunes (1502-1578) in Libro de Algebra en Arithmetica y Geometria, 1567, fols. 270–270v. ]
“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 resolver vna question por medios Geometricos, y resolverla por Algebra, siendo la misma Algebra sacada de la Geometria, ñ es cosa de admiraciô.

[ 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.

J. Backus.
Can programming be liberated from the von Neumann style? a functional style and its algebra of programs.

C.B. Jones.

*Development Methods for Computer Programs including a Notion of Interference.*
Printed as: Programming Research Group, Technical Monograph 25.

C.B. Jones.

*Systematic Software Development Using VDM.*
C.B. Jones.
Reasoning about partial functions in the formal development of programs.

J.N. Oliveira.
Draft of paper in preparation.

B. Schieder and M. Broy.
Adapting calculational logic to the undedfined.