Hands on a Grand Challenge in Computing: Proving a Journaled File System Correct

J.N. Oliveira

High Assurance Software Lab and Dept. Informática
Universidade do Minho
Braga, Portugal

INFORUM 2010
Braga, Portugal, September 2010

Opening questions

- Are we doing computer science research in the right way?
- Are we using the right notation, language?
- Does more technology mean better science?
- “Is computer science science?” (Denning, 2005)

Science? Pre-science?

In an excellent book on the history of scientific technology,

> *"How Science Was Born in 300BC and Why It Had to Be Reborn" (Springer, 2003),*

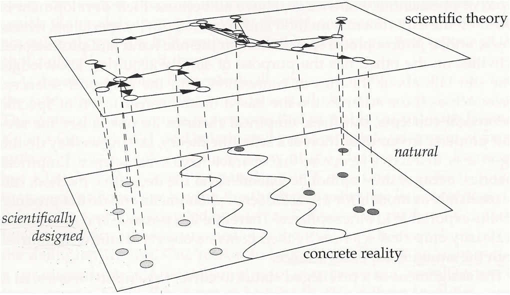
Lucio Russo writes:

> *The immense usefulness of* **exact** *science consists in providing* **models** *of the real world within which there is a guaranteed method for telling false statements from true. (...) Such models, of course, allow one to describe and* **predict** *natural phenomena, by translating them to the theoretical level via* **correspondence rules**, *then solving the "***exercises***" thus obtained and translating the solutions obtained back to the real world.*

Disciplines unable to build themselves around *"exercises"* are regarded as **pre-scientific**.

Scientific engineering ($e = m + c$)

Also from Russo's book :



Vertical lines mean **abstraction**, horizontal ones mean **calculation**:

engineering = model first, then calculate
($e = m + c$)

Theory? Practice?

Donald Knuth:

> *My experience has been that* ***theories*** *are often more structured and interesting when they are based on* ***real problems****; somehow they are more exciting than completely abstract theories will ever be.*

(Quoted from *The Dangers of Computer-science Theory*. Standford University, 1971)

This kind of position explains the **Grand Challenges in Computing** initiative.

Grand Challenge Initiative

- Healthy trend in **formal methods** (FM) research driven by the idea of a **Grand Challenge** (GC).
- Triggered by eminent computer scientists T. Hoare & J. Misra.
- VSTTE conference (*"Verified Software: Theories, Tools, Experiments"*) created as response to the challenge.
- VSTTE'05: Hoare proposes that time to start long term international cooperation research projects has arrived.
- Outcome to be gathered in a **Verified Software Repository** (VSR).
- No funding.

Verified Software Repository

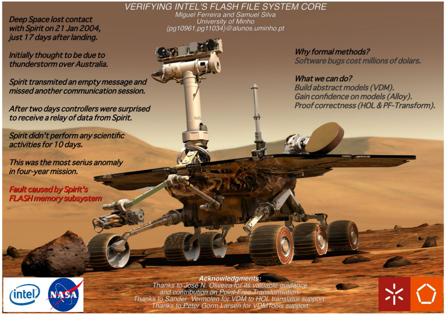
Mondex — A verified electronic purse hosted on a smart card. **Players**: Bremen (OCL); Escher Technologies (PerfectDeveloper); MIT (Alloy); Macao/DTU (Raise); Newcastle (p-Calculus); Southampton (Event-B); York (Z).

Pacemaker — based on a previous generation pacemaker specification released by Boston Scientific (BSC). Aims at production of verified pacemaker software, designed to run on specified PIC hardware. **Players** (thus far): Aharus (VDM++); BSC (BLESS); UFRGN (Z, PerfectDeveloper). UPEN (Uppaal, ADL).

Verified Software Repository

Verified File System (VFS) — Verified subset of POSIX suitable for flash-memory hardware with strict fault-tolerant requirements to be used by forthcoming NASA's JPL missions.

Players (thus far): Augsburg (KIV); MIT (Alloy); Minho (Alloy etc); Southampton (Event-B); York (Z/Eves).



VERIFYING INTEL'S FLASH FILE SYSTEM CORE

Miguel Ferreira and Samuel Silva
University of Minho
{pg10961,pg11034}@alunos.uminho.pt

Deep Space lost contact with Spirit on 21 Jan 2004, just 17 days after landing.

Initially thought to be due to thunderstorm over Australia.

Spirit transmited an empty message and missed another communication session.

After two days controllers were surprised to receive a relay of data from Spirit.

Spirit didn't perform any scientific activities for 10 days.

This was the most serius anomaly in four-year mission.

Fault caused by Spirit's FLASH memory subsystem

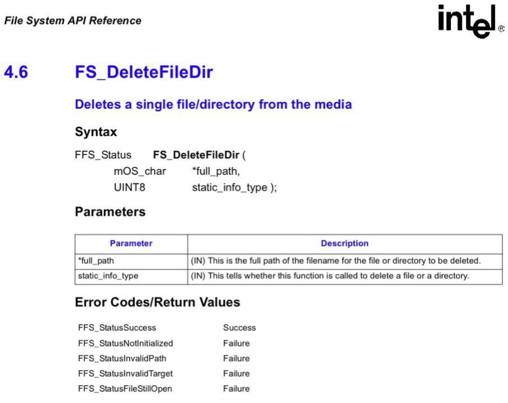
Why formal methods?
Software bugs cost millions of dolars.

What we can do?
Build abstract models (VDM).
Gain confidence on models (Alloy).
Proof correctness (HOL & PF-Transform).

Acknowledgments:
Thanks to José N. Oliveira for its valuable guidance
and contribution on Point-Free Transformation.
Thanks to Sander Vermolen for VDM to HOL translator support.
Thanks to Peter Gorm Larsen for VDMTools support.

VFS @ Minho

First effort was concerned with verifying Intel® *Flash File System Core Reference Guide* (API):



File System API Reference **intel®**

4.6 FS_DeleteFileDir

Deletes a single file/directory from the media

Syntax

```
FFS_Status    FS_DeleteFileDir (
          mOS_char      *full_path,
          UINT8         static_info_type );
```

Parameters

Parameter	Description
*full_path	(IN) This is the full path of the filename for the file or directory to be deleted.
static_info_type	(IN) This tells whether this function is called to delete a file or a directory.

Error Codes/Return Values

FFS_StatusSuccess	Success
FFS_StatusNotInitialized	Failure
FFS_StatusInvalidPath	Failure
FFS_StatusInvalidTarget	Failure
FFS_StatusFileStillOpen	Failure

(Permission to reproduce this excerpt kindly granted by Intel Corporation.)

VFS @ Minho

Formal model unveiled some ambiguities in the documentation, eg.

- Can the `root` directory be removed?

Surprised to see the POSIX System Interface Standard (2004) itself vague in this respect:

> *The rmdir() function shall remove a directory whose name is given by path. The directory shall be removed only if it is an empty directory. If the directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds, or whether it shall fail and set errno to [EBUSY].*

Publications: see (Oliveira, 2009), (Ferreira and Oliveira, 2009)

VFS @ Minho (recent)

GCI still suffering from lack of **comparative** work:

- We've chosen the KIV Augsburg model (Schierl et al., 2009) to compare our work with.
- Alloy emulation of Augsburg model — subject of a Master thesis by Fernandes (2010) available soon.
- Going abstract: high-level model in Alloy of the most interesting part of KIV model, which has to do with the **journaling**, **wear leveling** and **power loss** recovery mechanisms.
- Formal design and calculational approach (as explained later in this talk)

VFS @ Augsburg (KIV)

- **Standard:** They adhere to UBIFS (Unsorted Block Image File System) — a journaled file system developed by Nokia + Univ. Szeged that works on top of UBI (a wear-leveling and volume management system for flash devices).
- **Tool:** *Karlsruhe Interactive Verifier* (KIV) — a tactical theorem prover developed at the Univ. of Karlsruhe.

Main source: a nice paper

A. Schierl, G. Schellhorn, D. Haneberg, W. Reif *Abstract specification of the UBIFS file system for flash memory.* LNCS volume 5850, pages 190–206. Springer, 2009.

supported by a very detailed website:

`www.informatik.uni-augsburg.de/swt/projects/flash.html`

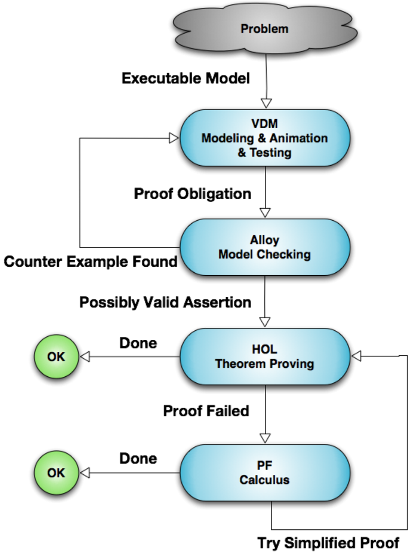
Our (first) approach

Verification life-cycle made of several steps:

- Build and animate the file system model (VDM++)
- Automatic generation of verification proof obligations (PO)
- PO model-checking step (Alloy)
- PO discharge step (HOL)

(Diagram next slide.)

Our (first) approach



What we have learnt

Mea culpa:

- Too technological
- Too many tools
- Tool interoperability at target in the GCI but hard to accomplish in practice
- Even if successful technology-wise: *"push-button proofs (alone) considered harmful"*
- Lack of *proof awareness* — proofs with too many (often hundreds, thousands) of steps.

Questions:

- How to reduce such complex models and proofs to something *small (readable) and elegant*?

Abstract modeling

There is a clear need for:

- **Abstraction**
- Notations in which you write **less** to say **more**
- **Calculi** to perform (readable) proofs as in high-school algebra.

My current answer to such needs is the

Relation algebra (RA)

which underlies the Algebra of Programming (Bird and de Moor, 1997). Why?

Abstract modeling

There is a clear need for:

- **Abstraction**
- Notations in which you write **less** to say **more**
- **Calculi** to perform (readable) proofs as in high-school algebra.

My current answer to such needs is the

Relation algebra (RA)

which underlies the Algebra of Programming (Bird and de Moor, 1997). Why?

Why Relation Algebra (RA)

- **Abstract models** capture nothing but the *essence* of given problems expressed in terms of **relationships** among **objects** of interest.
- So, **relational models** are natural and stem from **natural language** itself, cf. sentences such as eg.

  John loves Mary

  of the same shape as mathematical relationship

  $$0 \leq 1$$

  ("0 is at most 1"), and so on. (Note the **infix** notation.)

Why Relation Algebra (RA)

- The **algebra** of binary relations replaces logic deduction by inequational reasoning.
- Such calculations are **pointfree**, saving ink by dropping variables, quantifiers, variable substitution etc.
- Such was the motivation of mathematicians like Alfred Tarski (1901-83) who had a life-long struggle with quantified notation (too complex for his needs).

Why Relation Algebra (RA)

A bit of history:

- **RA** actually the first attempt to formal **knowledge representation**, carried out by Augustus de Morgan (1806-71) in his

  On the syllogism: IV, and on the logic of relations

  read on April 23, 1860 to the the Cambridge Philosophical Society. This **predates** logic itself.
- In fact, Peirce (1839-1914) invented *quantifier* notation to explain de Morgan's algebra of relations
- De Morgan's pioneering work was ill fated: the language invented to explain his **RA** became eventually more popular than **RA** itself.
- Tarski was among those who revived **RA**.

On the syllogism: IV (...)

Binary relations:

> [...] *Let $X..LY$ signify that X is some one of the objects of thought which stand to Y in the* **relation** *L, or is one of the Ls of Y.*

Relational composition:

> [...] *When the predicate is itself the subject of a relation, there may be a* **composition***: thus if $X..L(MY)$, if X be one of the Ls of one of the M s of Y, we may think of X as an 'L of M' of Y, expressed by $X..(LM)Y$, or simply by $X..LMY$.* [...][So] *brother of parent is identical with uncle, by mere definition.*

Relational converse:

> [...] *The* **converse** *relation of L, L^{-1}, is defined as usual: if $X..\ L\ Y$, $Y\ ..\ L^{-1}\ X$: if X be one of the Ls of Y, Y is one of the L^{-1} s of X.*

Binary relation essentials

Binary relations are **typed**:

Arrow $A \xrightarrow{R} B$ *denotes a binary relation from* A *(source) to* B *(target), where* A, B *are* **types**. *Writing* $B \xleftarrow{R} A$ *means the same as* $A \xrightarrow{R} B$.

Infix notation (such as verbs in natural language), eg.:

$$\mathtt{John}\ \mathit{Loves}\ \mathtt{Mary}$$

$$0 \ \leq \ \pi$$

$$b\ R\ a \qquad \text{(in general)}$$

(See Freyd and Scedrov (1990) for the foundations of typed **RA**.)

Composition and converse

Composition "... is R of some S of...":

$$\begin{tikzcd} B & A \arrow[l, "R"'] & C \arrow[l, "S"'] \arrow[ll, bend left, "R\cdot S"] \end{tikzcd}$$

$$b(R \cdot S)c \;\Leftrightarrow\; \langle \exists\, a \;::\; b\; R\; a \wedge a\; S\; c \rangle \tag{1}$$

Converse of R (or R in **"passive form"**): for all a, b,

$$\begin{tikzcd} A & B \arrow[l, "R^{\circ}"'] & & B & A \arrow[l, "R"'] \end{tikzcd}$$

$$a(R^{\circ})b \;\Leftrightarrow\; b\; R\; a \tag{2}$$

Note how (1) *removes* $\exists$ when applied from right to left.

Polymorphic relations

Top

$B \xleftarrow{\top} A$ *is the largest relation of its type —* $b \top a$ *always holds (this is de Morgan's* **"is coexistent with"** *relation)*

Bottom

$B \xleftarrow{\bot} A$ *is the smallest relation of its type —* $b \bot a$ *is always false*

Identity

$A \xleftarrow{id} A$ *is the smallest* **equivalence** *relation of its type —* $b\ id\ a$ *holds iff* $b = a$

The “at most” ordering

Each type $B \longleftarrow A$ forms a complete Boolean algebra (briefly speaking), whose ordering captures universal quantification:

$$R \subseteq S \quad \Leftrightarrow \quad \langle \forall\ b, a\ :\ b\ R\ a :\ b\ S\ a \rangle \tag{3}$$

Comments:

- Read $R \subseteq S$ as “R is at most S”, that is, “all Rs are Ss”.
- Note how (3) *removes* $\forall$ when applied from right to left. (“Pointfree” transform).
- What we have thus far is enough for much abstract modelling. . .
- Tarski would say: *“Isn’t [this] a nice thing ‘pour épater les logiciens-bourgeois’?”* (Givant, 2006)

Diagrams

Since relations are arrows, we can draw **diagrams** describing **constraints**, for instance

$$\begin{tikzcd}
Descriptor \arrow[d, "path"'] & & Handle \arrow[ll, "FT"'] \arrow[d, "\top"] \\
 & \subseteq & \\
Path & & File \arrow[ll, "FS^\circ"]
\end{tikzcd}$$

depicting constraint

$$path \cdot FT \subseteq FS^\circ \cdot \top \qquad (4)$$

where *FS* is a *file store*, *FT* is the open-file table and *path* yields the path of an open file descriptor. What does (4) mean, in predicate logic? See next slide.

From the "pointfree" (RA) to the "pointwise" (FOL)

We calculate:

$$
\begin{array}{ll}
 & path \cdot FT \subseteq FS^{\circ} \cdot \top \\
\Leftrightarrow & \qquad \{ \text{ `at most' ordering (3) } \} \\
 & \langle \forall\ p, h\ :\ p(path \cdot FT)h :\ p(FS^{\circ} \cdot \top)h \rangle \\
\Leftrightarrow & \qquad \{ \text{ composition (1) ; } path \text{ is a function } \} \\
 & \langle \forall\ p, h\ :\ \langle \exists\ d\ :\ p = path\ d :\ d\ FT\ h \rangle :\ p(FS^{\circ} \cdot \top)h \rangle \\
\Leftrightarrow & \qquad \{ \text{ quantifier calculus — splitting rule (Backhouse, 2003) } \} \\
 & \langle \forall\ d, h\ :\ d\ FT\ h :\ \langle \forall\ p\ :\ p = path\ d :\ p(FS^{\circ} \cdot \top)h \rangle \rangle \\
\Leftrightarrow & \qquad \{ \text{ quantifier calculus — one-point rule (Backhouse, 2003) } \} \\
 & \langle \forall\ d, h\ :\ d\ FT\ h :\ (path\ d)(FS^{\circ} \cdot \top)h \rangle
\end{array}
$$

From the “pointfree” (RA) to the “pointwise” (FOL)

We are left with $p(FS^\circ \cdot \top)h$:

$$
\begin{array}{ll}
 & p(FS^\circ \cdot \top)h \\
\Leftrightarrow & \qquad \{ \text{ composition again (1) } \} \\
 & \langle \exists\, x \;::\; p(FS^\circ)x \wedge x \top h \rangle \\
\Leftrightarrow & \qquad \{ \text{ converse ; } x \top h \text{ always holds } \} \\
 & \langle \exists\, x \;::\; x\ FS\ p \wedge \textsc{True} \rangle \\
\Leftrightarrow & \qquad \{ \text{ trivia } \} \\
 & \langle \exists\, x \;::\; x\ FS\ p \rangle
\end{array}
$$

Altogether, $path \cdot FT \subseteq FS^\circ \cdot \top$ unfolds into (next slide):

From the “pointfree” (RA) to the “pointwise” (FOL)

$$\langle \forall\ p, h\ :\ \langle \exists\ d\ ::\ d\ FT\ h\ \wedge\ p = path\ d \rangle : \langle \exists\ f\ ::\ f\ FS\ p \rangle \rangle$$

Informally:

$$\begin{tikzcd} d \arrow[d, mapsto, "path"'] \arrow[dr, phantom, "\subseteq"] & h \arrow[l, mapsto, "FT"'] \arrow[d, mapsto, "\top"] \\ p \arrow[r, mapsto, "FS"'] & f \end{tikzcd}$$

> *If h is the handle of a open-file descriptor d holding path p, then p points to some existing file f.*

In short:

> *Non-existing files cannot be opened. (***Referential integrity***)*

Model constraints are diagrams

Another example:

$$\begin{tikzcd}
\mathit{Path} \arrow[d, "\mathit{FS}"'] & \mathit{Path} \arrow[l, "\geq"'] \arrow[d, "\mathit{FS}"] \arrow[dl, phantom, "\subseteq"] \\
\mathit{File} & \mathit{File} \arrow[l, "\top"]
\end{tikzcd}$$

that is $FS \cdot \geq \subseteq \top \cdot FS$

where $p \leq p'$ means p *is a sub-path of* p' and whose meaning is

> *Mother-directories always exist.*

Summary:

> *Properties such as* **referential integrity**, *prefix-closure and many others are captured by easy-to-grasp* **RA** *expressions depicted by diagrams.*

Model constraints are diagrams

Other examples (in general):

- M, N domain-disjoint:

$$M \cdot N^{\circ} \quad \subseteq \quad \bot$$

- M, N domain-coherent:

$$M \cdot N^{\circ} \quad \subseteq \quad id$$

- Domain of M strictly above that of N:

$$M^{\circ} \cdot \top \cdot N \quad \subseteq \quad >$$

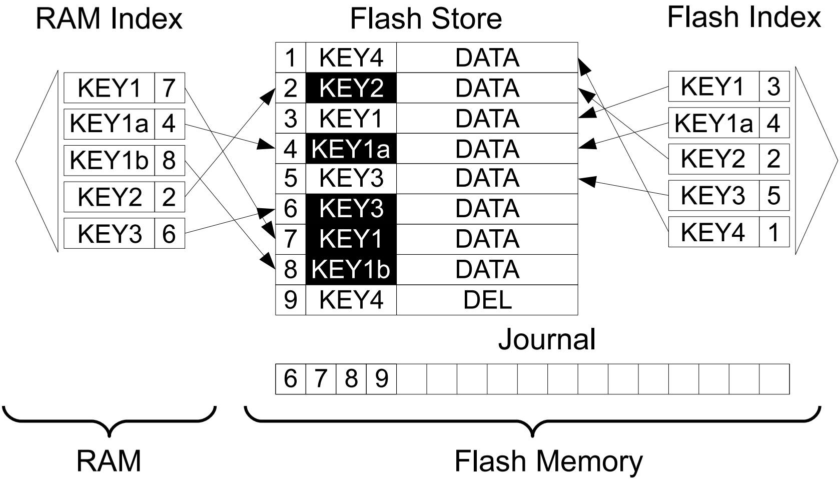
Flash file store refinement

The model of a flash-memory file-store is far more complex than what has been hinted above, given extra non-functional requirements such as:

- **Performance:** core meta-data is stored in central memory (RAM) to decrease update latency.
- **Wear leveling:** no delete/write cycles so as to prolong the service life of this erasable storage media.
- **Power-loss recovery:** to add robustness to the system against faults of this kind, a *backlog* (**journal**) of the operations that have been performed is stored in the device.

Journaled FLASH store snapshot

In a picture, quoted from Schierl et al. (2009):



RAM Index

KEY1	7
KEY1a	4
KEY1b	8
KEY2	2
KEY3	6

Flash Store

1	KEY4	DATA
2	KEY2	DATA
3	KEY1	DATA
4	KEY1a	DATA
5	KEY3	DATA
6	KEY3	DATA
7	KEY1	DATA
8	KEY1b	DATA
9	KEY4	DEL

Flash Index

KEY1	3
KEY1a	4
KEY2	2
KEY3	5
KEY4	1

Journal

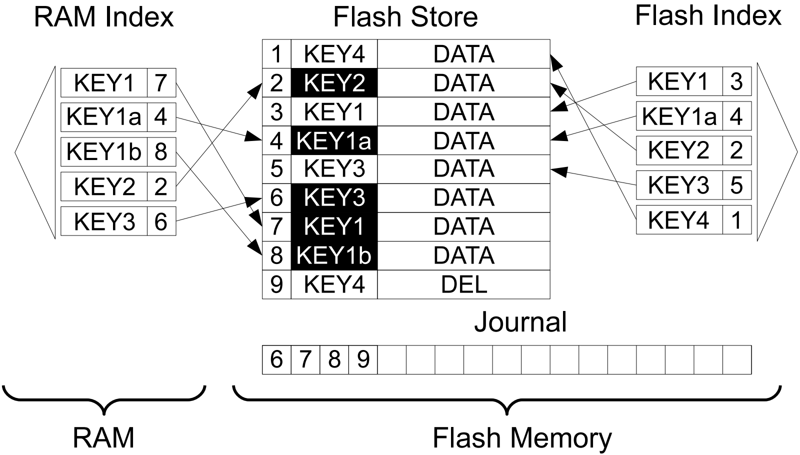
6	7	8	9													

RAM

Flash Memory

Journaled file store snapshot

Note how **updating** (eg. key *KEY*3) and **deletion** (eg. key *KEY*4) actually entail **new** entries in the FLASH (thus the counter-intuitive fact that deletion calls for extra free space):



RAM Index

KEY1	7
KEY1a	4
KEY1b	8
KEY2	2
KEY3	6

Flash Store

1	KEY4	DATA
2	KEY2	DATA
3	KEY1	DATA
4	KEY1a	DATA
5	KEY3	DATA
6	KEY3	DATA
7	KEY1	DATA
8	KEY1b	DATA
9	KEY4	DEL

Flash Index

KEY1	3
KEY1a	4
KEY2	2
KEY3	5
KEY4	1

Journal

6	7	8	9													

FLASH abstract model in a diagram

Generic model

$$\begin{tikzcd}
 & & K \arrow[dl, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd} \tag{5}$$

where A (memory addresses), K (keys) and D (data) can be regarded at any level — (eg. $K = Path$, $K = inode\ number$, etc).

Abstract model in a diagram

$$\begin{tikzcd}
 & & K \arrow[dl, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Concrete states:

FS — FLASH store

RI — RAM index

FI — FLASH index

J — Journal (sequence of addresses)

Abstract states:

R — Abstract K-D relationship being implemented.

Abstract model in a diagram

$$\begin{tikzcd}
 & & K \arrow[dl, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Data types:

$D+1$ — accommodates both valid data (D) and the *DEL* mark (1) intended for recording data deletion. Membership relation $d \in x$ picks data from a non-*DEL* entry x.

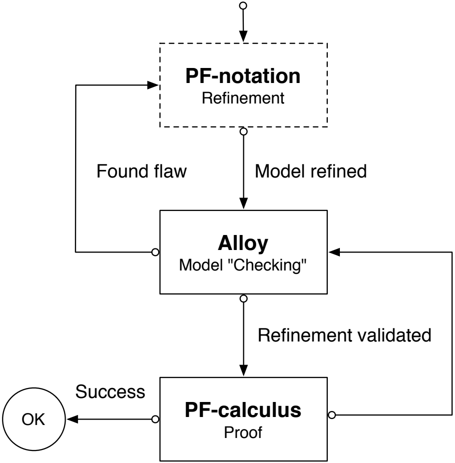
$K \times (D+1)$ — accommodates pairs (k, d) of keys and (maybe) data; projections π_1 and π_2 such that $\pi_1(k, d) = k$ and $\pi_2(k, d) = d$.

Alloy

Bringing Alloy in — why?

- Need to obtain **feedback** about our model
- Need to know as soon as possible if building an **inconsistent** model
- **Counter-examples** very useful in case of nonsense proof-obligations
- Alloy's **pointfree** subset is very close to **RA**

Thus the *minimalist* **verification life-cycle** on the right.



Abstract model diagram in Alloy

$$\begin{tikzcd}
 & & K \arrow[dl, "{FI,RI}"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"'] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Abstract states:

```
sig AS { r : K → lone D }
```

Concrete states:

```
sig CS {
  j : N → lone A,
  fs : A → lone Entry,
  fi : K → lone A,
  ri : K → lone A
}
```

Abstract model diagram in Alloy

$$\begin{tikzcd}
 & & K \arrow[lld, "{FI,RI}"'] \arrow[rdd, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"'] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

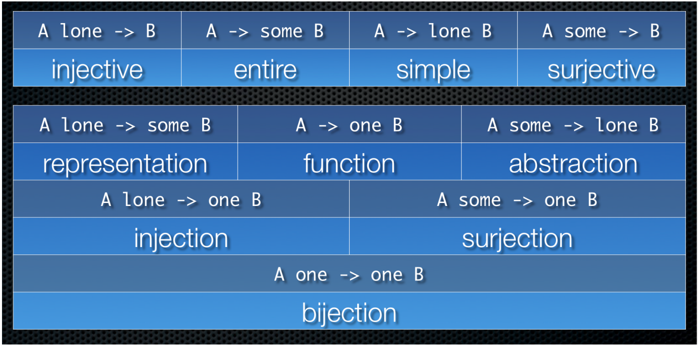
$K \times (D+1)$

```
sig Entry { key : one K, value : one DataCell }
```

$D+1$

```
abstract sig DataCell {}
one sig Del extends DataCell {}
sig Data extends DataCell { data: one D }
```

Multiplicities in Alloy + taxonomy



<table>
<tr><td colspan="3"><code>A lone -> B</code></td><td colspan="3"><code>A -> some B</code></td><td colspan="3"><code>A -> lone B</code></td><td colspan="3"><code>A some -> B</code></td></tr>
<tr><td colspan="3">injective</td><td colspan="3">entire</td><td colspan="3">simple</td><td colspan="3">surjective</td></tr>
<tr><td colspan="4"><code>A lone -> some B</code></td><td colspan="4"><code>A -> one B</code></td><td colspan="4"><code>A some -> lone B</code></td></tr>
<tr><td colspan="4">representation</td><td colspan="4">function</td><td colspan="4">abstraction</td></tr>
<tr><td colspan="6"><code>A lone -> one B</code></td><td colspan="6"><code>A some -> one B</code></td></tr>
<tr><td colspan="6">injection</td><td colspan="6">surjection</td></tr>
<tr><td colspan="12"><code>A one -> one B</code></td></tr>
<tr><td colspan="12">bijection</td></tr>
</table>

where

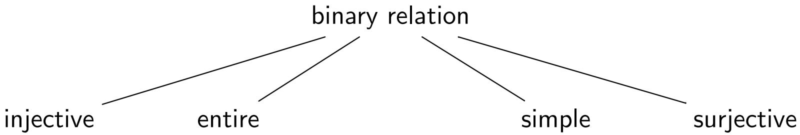
lone — at most one

some — at least one

one — exactly one

The same — mathematically

Topmost criteria:



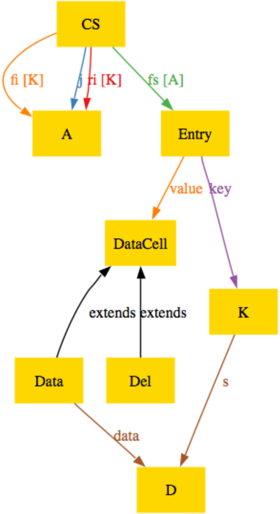
Definitions:

	$\supseteq id$	$\subseteq id$
$\ker R$	entire R	injective R
$\operatorname{img} R$	surjective R	simple R

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

$$\operatorname{img} R = R \cdot R^{\circ}$$

Abstract model as depicted by Alloy



Abstraction invariant

$$\begin{tikzcd}
& & K \arrow[dl, "FI,RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"] & \\
& & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

$$R \quad = \quad af(FS, FI, RI, J)$$

where abstraction function is

$$af(FS, FI, RI, J) \quad \triangleq \quad (active\ \ FS) \cdot RI \tag{6}$$

for

$$A \xrightarrow{active\ \ FS} D \quad \triangleq \quad \in \cdot \pi_2 \cdot FS \tag{7}$$

Abstraction invariant in Alloy

$$\begin{tikzcd}
 & & K \arrow[ld, "{FI,RI}"'] \arrow[rdd, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"'] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Abstraction function $af(FS, FI, RI, J) \triangleq (active\ FS) \cdot RI$

```
fun af[cs: CS] : K → D { (cs·ri)·(cs·fs·active) }
```

Auxiliary function $active\ FS \triangleq \in \cdot \pi_2 \cdot FS$

```
fun active[x: CS·fs] : A → D { x·value·data }
```

(Mind that **reverse** order in which Alloy chains the arguments of relational **composition**.)

Concrete invariant

Recall the class of simple relations (partial functions), where $A \xrightarrow{S} B$ is simple iff $S \cdot S^\circ \subseteq id$ which, once variables are added, means

$$\langle \forall\ b, b' \ :\ \langle \exists\ a \ ::\ bSa \wedge b'Sa \rangle :\ b = b' \rangle$$

($=S$ is univocal, deterministic). Clearly, we want FS, J, RI, FI simple.

$$\begin{tikzcd}
 & & K \arrow[dll, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi1"'] \arrow[d, "\pi2"] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Concrete invariant

Recall FLASH non-functional requirements such as **wear leveling** and **power loss** recovery. In the event of a power loss RI will be lost. The redundancy of FS (5) is intended for recovery, provided **consistency** clause

$$RI \ \subseteq \ index\ FS \tag{8}$$

is added to the concrete invariant, where

$$K \xrightarrow{index\ \ FS} A \quad \triangleq \quad (\pi_1 \cdot FS)^\circ \tag{9}$$

```
fun index[x: CS·fs] : K → A { ˜(x·key) }
```

$$\begin{tikzcd}
 & & K \arrow[dll, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Concrete invariant

Recovery possible only if there is a function which **rebuilds** RI from the other, persistent (FLASH-stored) relations. First attempt:

$$RI \;=\; index\ FS$$

Doesn't work! $index\ FS = (\pi_1 \cdot FS)^\circ$ is in general injective but not simple. Why? Keeping track of all addresses which have been involved in recording data for a given k, information is missing about which addresses correspond to the **most recent** updates.

$$\begin{tikzcd}
 & & K \arrow[dl, "FI{,}RI"'] \arrow[ddr, "R"] & \\
\mathbb{N} \arrow[r, "J"] & A \arrow[r, "FS"] & K \times (D+1) \arrow[u, "\pi 1"'] \arrow[d, "\pi 2"] & \\
 & & (D+1) \arrow[r, "\in"] & D
\end{tikzcd}$$

Rules of thumb:

1. R° simple iff R injective
2. R° entire iff R surjective

Bringing journal into the play

We need to reduce the non-determinism of *index FS* by selecting only the most recent updates. This is where the **journal** helps,

$$A \xleftarrow{\geq_J} A \quad \triangleq \quad J \cdot \geq \cdot J^{\circ} \tag{10}$$

ordering addresses by comparing their relative positions in J, larger positions meaning more recent updates:

$$a \geq_J b \quad \Leftrightarrow \quad \langle \exists\ t, t' \ :: \ a\ J\ t \wedge b\ J\ t' \wedge t \ \geq \ t' \rangle$$

Then we use the relational **"shrink" combinator** to express the selection of the most recent update per key:

$$RI \quad = \quad index\ FS \upharpoonright (\geq_J)$$

Let us explain what it means.

Intuition about $R \upharpoonright S$

Example of $R \upharpoonright S$ in data-processing:

$$\left(\begin{array}{c|c} Mark & Student \\ \hline 10 & John \\ 11 & Mary \\ 12 & John \\ 15 & Arthur \end{array}\right) \upharpoonright \geq \quad = \quad \begin{array}{c|c} Mark & Student \\ \hline 11 & Mary \\ 12 & John \\ 15 & Arthur \end{array}$$

Example of $R \upharpoonright S$ in list-processing: given a sequence $\mathbb{N} \xrightarrow{S} A$,

$$\mathbb{N} \xrightarrow{nub\ S} A \quad \triangleq \quad (S^\circ \upharpoonright \leq)^\circ$$

removes all duplicates while keeping the first instances. ($\mathbb{N}$ could be regarded as a time stamp.)

Definition of $R \upharpoonright S$

Given relation $B \xleftarrow{R} A$ and optimization criterion $B \xleftarrow{S} B$ on its outputs,

$$\begin{tikzcd}
& A \arrow[dl, "R\upharpoonright S"'] \arrow[d, "R"] \\
B & B \arrow[l, "S"]
\end{tikzcd}$$

define $R \upharpoonright S$ satisfying universal property:

$$X \subseteq R \upharpoonright S \quad \Leftrightarrow \quad X \subseteq R \ \wedge \ X \cdot R^{\circ} \subseteq S \tag{11}$$

This ensures $R \upharpoonright S$ as the largest sub-relation X of R such that, for all $b', b \in B$, if there exists $a \in A$ such that $b' X a \wedge b R a$, then $b' S b$ holds ("b' better than b").

$$\begin{tikzcd}
& a \arrow[dl, mapsto, "X"'] \arrow[d, mapsto, "R"] \\
b' & b \arrow[l, mapsto, "S"]
\end{tikzcd}$$

Algebra of $R \upharpoonright S$

Chaotic optimization:

$$R \upharpoonright \top \quad = \quad R \tag{12}$$

Impossible optimization:

$$R \upharpoonright \bot \quad = \quad \bot \tag{13}$$

Ensure simplicity (determinism):

$$R \upharpoonright S \text{ is simple } \Leftarrow S \text{ is anti-symmetric} \tag{14}$$

Select determinism:

$$R \upharpoonright id \quad = \quad \text{largest deterministic fragment of } R \tag{15}$$

Gaining what?

- No (explicit) recursion. (Hidden in the $R \upharpoonright S$ combinator.)
- No need for inductive proofs.
- Deductive calculation as in high school algebra, thanks to distributive, permutative properties, eg.

$$(R \cup S) \upharpoonright U = (R \upharpoonright U) \cup (S \upharpoonright U) \ \Leftarrow \ R \cdot S^{\circ} \subseteq \bot$$

- Theory reusable elsewhere (eg. currently using $R \upharpoonright S$ in calculating **greedy algorithms** from specifications expressed by Galois connections).

Back to VFS — the *replay* function

Finally, power-loss recovery is performed by the so-called *replay* function,

$$replay(FS, FI, J) \quad \triangleq \quad (active\ FS) \lhd (index\ FS \restriction (\geq_J))$$

where there is a final stage of filtering deleted keys out resorting to another combinator

$$S \lhd R \triangleq \ S^{\circ} \cdot \top \cap R$$

which picks that part of R which "chains" with S. (Read $S \lhd R$ as "R if S is defined"; $\cap$ denotes relation intersection.)

$$\begin{tikzcd}
 & K \arrow[dl, "index\ FS \restriction \geq_J"'] \arrow[d, "index\ FS"] \\
A \arrow[dr, "active\ FS"'] & A \arrow[l, "\geq_J"] \\
 & D
\end{tikzcd}$$

Concrete invariant

Ensures that, at any time, *replay* recovers **the current** RI,

$$\begin{array}{rcl} ci(FS, FI, RI, J) & \triangleq & RI = replay(FS, FI, J) \ \wedge \\ & & J \text{ is injective } \ \wedge \\ & & eqdef(J^{\circ}, FS) \end{array}$$

where

- J injective ensures $\geq_J$ anti-symmetric (cf. RI deterministic);
- $eqdef(R, S)$ ensures that R and S are equally defined.

The last requirement is still too strong: J is bound to cover the whole FS at any time and thus power-loss recovery of RI by the *replay* function will thus take longer and longer as J grows.

FLASH index (FI) gets into the play

- From time to time, J should be cleared up while saving the contents of RI persistently.
- Such is the purpose of FI (flash index), a component of the state model which has played no role in the model so far.
- In this way, J will keep only the "difference" between the RI and its *cache* FI, very often outdated.
- Introducing FI requires a *commit* operation which basically updates the FI with the contents of RI and clears J, as specified by post-condition:

$$
\begin{aligned}
&J' = \perp\\
&FI' = RI\\
&FS \text{ remains unchanged}\\
&RI \text{ remains unchanged}
\end{aligned}
$$

Concrete invariant (final version)

Caching *RI* into *FI* adds further complexity to the *replay* operation,

$$replay(FS, FI, J) \quad = \quad (active\ FS) \lhd (FI \dagger ((J^\circ \lhd (index\ FS)) \upharpoonright (\geq_J)))$$

— where $\dagger$ denotes relational overriding — but has the advantage of replaying only the operations that happened after the last check point (*commit*).

Pragmatics: need for extra term $J^\circ \lhd$ in the definition is quite subtle: it was prompted to us by a painful counter-example generated by the Alloy model-checker.

Device full (garbage collection)

Specified by post-condition,

$$\begin{array}{l} J, FI, RI \text{ remains unchanged} \\ FS' = FS \cap \top \cdot RI^{\circ} \end{array}$$

this operation reclaims all FS entries inaccessible to RI, ie. those which mark deletions or outdated information, consequence of the **wear-levelling** principle. With points, the garbage-collected flash store is such that

$$x \; FS' a \;\Leftrightarrow\; x \; FS \; a \wedge \langle \exists \; k \;::\; a \; RI \; k \rangle$$

Wear-leveling's implications in the model are clearly shown by the complexity of one of the usually most simple CRUD operations — *deletion* — to be given next.

Delete operation

Alloy:

```
pred Delete[cs,cs’: CS, s: set K] {
  some n: K → lone A, m: N → lone A {
    injective[n, A] and injective[m, A]
    no n·(cs·fs) and n·ran = m·ran
    n·dom = s and cs·j·Top·(˜m) in ˆ(ordering/prev)

    cs’·j = cs·j + m
    cs’·fs = cs·fs + n·del
    cs’·ri = (cs·fs·dom − s)·(cs·ri)
    cs’·fi = cs·fi
  }
}
```

where

```
fun del[n: K → A] : A → Entry
    { { a: n·ran, e: Entry | e·key in n·dom and e·value = DEL } }
```

Verification

Life-cycle:

- This generates (among others) proof obligation:

```
assert po47 {
  all cs,cs': CS, s : set K |
    (ci[cs] and Delete[cs,cs',s]) ⇒ cs'·ri = cs'·replay
}
```

- Alloy finds no counter-examples
- We thus proceed to manual proof (next slide)

Calculation

Nine (deductive) steps:

$$
\begin{array}{ll}
 & \mathit{replay}(\mathit{FS}', \mathit{FI}', J') \\
= & \quad \{ \text{ substitutions enabled by post-condition } \} \\
 & \mathit{replay}(\mathit{FS} \cup \mathit{del}\ N, \mathit{FI}, J') \\
= & \quad \{ \text{ definition of } \mathit{replay} \text{ ; } \mathit{active} \text{ distributes over union } \} \\
 & (\mathit{active}\ \mathit{FS} \cup \mathit{active}(\mathit{del}\ N)) \lhd (\mathit{index}\ (\mathit{FS} \cup \mathit{del}\ N) \restriction (\geq_{J'})) \\
= & \quad \{ \text{ orthogonality ; } \mathit{index} \text{ distributes over union ; } \mathit{del} \ \} \\
 & (\mathit{active}\ \mathit{FS}) \lhd ((\mathit{index}\ \mathit{FS} \cup N) \restriction (\geq_{J'})) \\
\vdots & \quad \{ \text{ 5 steps omitted for presentation purposes } \} \\
 & \mathit{RI} \cdot (\notin S) \cup \bot \\
= & \quad \{ \text{ post-condition } \} \\
 & \mathit{RI}'
\end{array}
$$

On the RA-Alloy interplay

Comment by a referee:

> *(...) difficult proof steps model-checked first ... seems like an excellent idea, but how do you know when to model-check, wait until your proof is running into trouble? If this is still a matter of good mathematical judgement, this should be made clear.*

Our answer is a quite pragmatic design principle:

> *Whatever you are going to do in applied* **RA***, model-check it first. Silly errors are very likely in complex designs — even using* **RA :-)**

(Recall design of final version of *replay*.)

Lessons learnt

- Smart notation better than heavy machinery
- Abstraction as main ingredient of scientific engineering
- GCI the right way to go:
  - builds communities
  - enables comparative work
  - pushes you favourite method to its limits
  - has social impact

If you know of a GC in our area of work — just embrace it!

What next

From (another) referee:

> *(...) engineering practitioners have mathematical maturity but cannot be expected to be widely educated on abstract mathematics. (...)*
> *The authors need to plan on writing a textbook that explains the approach, in much the same manner as the various textbooks for Z, starting from first principles (ZF set theory and first order logic), if they really expect the approach to see significant industrial use.*

What next

- Fine — the book is (slowly) under way. . .
- Meanwhile, teaching experience has been gathered in the last three years in teaching **RA** (with applications) to MSc students, as a module of the MFES ("Métodos Formais em Engenharia de Software") unit — slides available from

  `wiki.di.uminho.pt/twiki/bin/view/Education/MFES/`
- Interested in **relational thinking**? Join the club — you are already 150 years late! (2010-1860=150)

Questions and answers

Question: do you advocate returning to manual proofs, full stop?

> **Answer:** *No — the size and complexity of today's problems make hand-proving alone unrealistic. What is advocated is* ***relational thinking****, a change in the way we think about software.*

Question: why Alloy? Couldn't other model-checkers be used instead?

> **Answer:** *Hmm... perhaps not — Alloy's main inspiration is Tarski's stuff (much more relevant than the Z inspiration). In particular, Alloy's pointfree subset is very close do* ***RA****.*

Questions and answers

Question: how reliable (formal) is your translation between **RA** and Alloy?

> **Answer:** *Manual but systematic; a translation tool has just become available* **:-)**
>
> N. Macedo *Translating Alloy specifications to the point-free style. Master's thesis, Minho University, 2010. (Submitted.)*

Question: does theorem proving (TP) still find a place in your approach?

> **Answer:** *Of course it does! But the level of proofs needs to raise up to* **RA***-styled proofs. Höfner and Struth (2008) show that TP such as* **Prover9** *already work at this level.*

Questions and answers

Question: is **RA** expressive enough to tackle “big” problems such as those of the GCI?

> **Answer:** *Yes – and it gives you room to “invent” new combinators (such as $R \upharpoonright S$) and exploit their algebra — this saves much work and structures the reasoning.*

Question: what is the relationship between **RA** and the database homonym “à la Codd”?

> **Answer:** *Codd’s multi-ary relation theory instantiates* **RA***, once its set-theoretic foundations are “pointfreed”*

Question: can other families of logics, for instance **temporal logic** be pointfreed in **RA**?

> **Answer:** *Yes – Raymond Boute (2009) got a best paper award at FM’09 (Eindhoven World Congress) doing so.*

Questions and answers

Question: what about probabilistic modelling, Markov chains and the like?

Answer: ***RA*** *is in many respects a linear-algebra (****LA****), as binary relations are just Boolean matrices. Composition instantiates matrix multiplication and converse matrix transposition. So* ***RA*** *is much closer to* ***LA*** *than predicate logic.*

Question: you rely much on diagrams; could UML diagrams be used instead?

Answer: *UML diagrams are informal and thus hard to reason about; our diagrams are central to the underlying* ***allegory*** *theory — they can even be used as proofs (Freyd and Scedrov, 1990).*

References

Roland Backhouse. *Program Construction: Calculating Implementations from Specifications*. John Wiley & Sons, Inc., New York, NY, USA, 2003. ISBN 0470848820.

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

Raymond Boute. Making temporal logic calculational: A tool for unification and discovery. In *FM'09: Proceedings of the 2nd World Congress on Formal Methods*, pages 387–402, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05088-6. doi: http://dx.doi.org/10.1007/978-3-642-05089-3_25.

Peter J. Denning. Is computer science science? *Commun. ACM*, 48(4):27–31, 2005.

M.J. Fernandes. Formal verification using the esc-pf calculus and model-checking in alloy of the ubifs file system for flash memory. Master's thesis, University of Minho, Informatics Department, 2010. In preparation.

M.A. Ferreira and J.N. Oliveira. An Integrated Formal Methods Tool-Chain and Its Application to Verifying a File System

Model. In *SBMF*, volume 5902 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 2009. (Best paper award).

P.J. Freyd and A. Scedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.

S. Givant. The calculus of relations as a foundation for mathematics. *J. Autom. Reasoning*, 37(4):277–322, 2006. ISSN 0168-7433. doi: http://dx.doi.org/10.1007/s10817-006-9062-x.

Peter Höfner and Georg Struth. On automating the calculus of relations. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2008. ISBN 978-3-540-71069-1.

J.N. Oliveira. Extended Static Checking by Calculation using the Pointfree Transform . In A. Bove et al., editor, *LerNet ALFA Summer School 2008*, volume 5520 of *LNCS*, pages 195–251. Springer-Verlag, 2009.

L. Russo. *The Forgotten Revolution: How Science Was Born in 300BC and Why It Had to Be Reborn*. Springer-Verlag, September 2003. URL `http://www.springer.com/978-3-540-20396-4`.

Andreas Schierl, Gerhard Schellhorn, Dominik Haneberg, and Wolfgang Reif. Abstract specification of the UBIFS file system for flash memory. In Ana Cavalcanti and Dennis Dams, editors, *FM'09*, volume 5850 of *Lecture Notes in Computer Science*, pages 190–206. Springer, 2009. ISBN 978-3-642-05088-6.

Open Group Technical Standard. Standard for information technology - Portable operating system interface (POSIX). System interfaces. *IEEE Std 1003.1, 2004 Edition. The Open Group Technical Standard. Base Specifications, Issue 6. Includes IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. System Interfaces*, 2004.