Theory and applications of the PF-transform

J.N. Oliveira

Dept. Informática,
Univ. Minho
Braga, Portugal

LERNET School 2008, Piriápolis, Uruguay
(Last update: January 2009)

Syllabus

5 lectures (50m each):

- First lecture (Mon 25 Feb)

  Introduction and motivation. Rigorous software development: the $e = m + c$ equation. Description versus calculation. Pointwise versus pointfree notation. Software properties as quantified formulæ. Eindhoven quantifier calculus.

- Second lecture (Mon 25 Feb)

  PF-transform essentials. Binary relation combinators. Rules of the PF-transform. The role of composition. Taxonomy of binary relations. Functions. "Al-djabr" rules.

Syllabus

- Third lecture (Tue 26 Feb)

  Data-type invariants. PF-transform of unary predicates. Coreflexives and conditions. Proof obligations (PO): invariant preservation. PF-transformed POs. Relation to Hoare logic. Using the Alloy Analyser as a PF-transform checker.

- Fourth lecture (Tue 26 Feb)

  Discharging proof obligations via PF-transform. Pre/post conditions. Invariants. Extended static checking in the PF-style. PF-calculation of weakest pre-conditions for invariant preservation.

Syllabus

- Fifth lecture (Wed 27 Feb)

  Proof obligations in-the-large and in-the-small. Thinking big writing less. The VFS (Verified File System) case study. The broad picture: integration with theorem provers and model checkers The broad picture: invariants as coreflexive bisimulations in a coalgebraic setting.

First lecture

Schedule: Monday Feb 25th, 16h20-17h10

Learning outcomes:

- Identifying the **problem**
- Finding a **strategy** to face it
- Why the **PF-transform**

Motivation

- Much of our **effort** in programming goes into making sure that a number of ("good") **relationships** hold among the **artifacts** we build.

- We have two main ways of **ensuring** that such *good things* happen:

  - **postulate** the relationship + **verify** what has been postulated ("invent & verify")
  - **build** the relationship out of existing valid relationships using an **algebra** of relationships ("correct by construction")

Example — type checking

In functional programming, eg. Haskell:

- **Postulate**:

$$\underbrace{f}_{function} \quad :: \quad \underbrace{a \rightarrow b}_{type}$$

- **Artifacts**: functions (λ-expressions), types (τ-expressions)
- **Relationship**: *"is of type"*
- **Invent & verify**: declare $f :: a \rightarrow b$, define f and wait for the interpreter's reaction
- **Correct by construction**: start by defining f, then let the interpreter calculate its (principal) type; instantiate this if required.

Example — Hoare logic

- **Postulate**:

$$\{p\}P\{q\}$$

— in fact "the same" as

$$\underbrace{P}_{program} \quad :: \quad \underbrace{p \rightarrow q}_{predicative\ type}$$

- **Artifacts**: programs (imperative code), pre/post assertions (predicates)
- **Relationship**: *"such that pre-condition p ensures post-condition q"*
- **Invent & verify**: write P, invent p and q and prove that $\{p\}P\{q\}$ holds
- **Correct by construction**: write q and P; calculate the **wp** for q to hold upon execution of P; obtain p by going stronger, if required.

Example — Refining specifications

- **Postulate**:

$$\underbrace{P}_{\textit{program}} \quad \sqsupseteq \quad \underbrace{S}_{\textit{specification}}$$

- **Artifacts**: programs, specifications
- **Relationship**: *"is a correct implementation of"*
- **Invent & verify**: given S, invent P and then prove that the semantics of P are more defined than S
- **Correct by construction**: calculate P by transforming S according to some refinement algebra compatible with $\sqsubseteq$.

Example — in discrete maths

- **Postulate**:

function f is a bijection

- **Artifacts**: functions, isomorphisms etc
- **Invent & verify**: given f, invent its converse f° and then prove the two cancellations

$$\langle \forall\ x\ ::\ f^\circ(f(x)) = x \rangle$$
$$\langle \forall\ y\ ::\ f(f^\circ(y)) = y \rangle$$

- **Correct by construction**: from f **calculate** f° (which in general is not a function); both f and f° will be bijective iff a function f° is obtained.

Our Aims

Aim:

- Our lectures will be devoted to the **calculational**, constructive option illustrated above

However:

- "Traditional" reasoning follows *invent & verify*
- Thinking constructively requires a "turn of mind"

Question:

- Are the logics and calculi we traditionally rely upon up-to-date for such a turn of mind ?

Scientific? Pre-scientific?

In an excellent essay on the history of scientific technology, Russo [13] 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**.

Rigorous software development

Adopting a **formal** notation instead of some programming notation (language) doesn't mean by itself that one is following a formal approach:

- formal 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...

Rigorous software development

Adopting a **formal** notation instead of some programming notation (language) doesn't mean by itself that one is following a formal approach:

- formal 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...

Rigorous software development

Adopting a **formal** notation instead of some programming notation (language) doesn't mean by itself that one is following a formal approach:

- formal 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...

Problem-solving strategy

Recall the *universal problem solving* strategy which one is taught at school:

- **understand** your problem
- build a mathematical **model** of it
- **reason** in such a model
- upgrade your model, if necessary
- **calculate** a final solution and implement it.

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{array}{cl} & 3x + 9 = 48 \\ \Leftrightarrow & \{ \text{ “al-djabr” rule } \} \\ & 3x = 48 - 9 \\ \Leftrightarrow & \{ \text{ “al-hatt” rule } \} \\ & x = 16 - 3 \end{array}$$

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{array}{cl} & 3x + 9 = 48 \\ \Leftrightarrow & \{ \text{ "al-djabr" rule } \} \\ & 3x = 48 - 9 \\ \Leftrightarrow & \{ \text{ "al-hatt" rule } \} \\ & x = 16 - 3 \end{array}$$

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{aligned}
& 3x + 9 = 48 \\
\Leftrightarrow \quad & \{ \text{ "al-djabr" rule } \} \\
& 3x = 48 - 9 \\
\Leftrightarrow \quad & \{ \text{ "al-hatt" rule } \} \\
& x = 16 - 3
\end{aligned}$$

School maths example

The solution

$$\begin{aligned} x &= 13 \\ x+3 &= 16 \\ x+6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Rules known since *On the calculus of al-gabr and al-muqâbala* by Abû **Al-Huwârizmî**, the famous 9c Persian mathematician.

School maths example

The solution

$$\begin{aligned} x &= 13 \\ x+3 &= 16 \\ x+6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Rules known since *On the calculus of al-gabr and al-muqâbala* by Abû **Al-Huwârizmî**, the famous 9c Persian mathematician.

School maths example

The solution

$$\begin{aligned} x &= 13 \\ x+3 &= 16 \\ x+6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Rules known since *On the calculus of al-gabr and al-muqâbala* by Abû **Al-Huwârizmî**, the famous 9c Persian mathematician.

Calculus of al-djabr, al-hatt and al-muqâbala

al-djabr

$$x - z \leq y \quad \Leftrightarrow \quad x \leq y + z$$

al-hatt

$$x * z \leq y \quad \Leftrightarrow \quad x \leq y * z^{-1} \qquad (z > 0)$$

al-muqâbala

Ex: $4x^2 - 2x^2 = 2x + 6 - 3 \quad \Leftrightarrow \quad 2x^2 = 2x + 3$

"Al-djabr" rules are not a privilege of arithmetics

- For instance, in predicate logic:

$$(x \wedge \neg z) \Rightarrow y \quad \Leftrightarrow \quad x \Rightarrow (z \vee y) \tag{1}$$

$$(x \wedge z) \Rightarrow y \quad \Leftrightarrow \quad x \Rightarrow (z \Rightarrow y) \tag{2}$$

  hold, for all x, y and z.

- "Al-djabr" rules are nowadays known as **Galois connections**.
- Can our reasoning in **rigorous software development** be performed with a similar degree of calculational accuracy and elegance?
- First of all: what **kind** of problems do we want to be rigorous about?

Software design (toy) example

The problem

Requirements fragment:

> *(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.*

The model

$$store\ c \quad \triangleq \quad (take\ 10) \cdot (c :) \cdot filter(c \neq) \tag{3}$$

where *take* and *filter* are the obvious functions.

The calculation

You said what...? 

Software design (toy) example

The problem

Requirements fragment:

> *(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.*

The model

$$\mathit{store}\ c \ \triangleq\ (\mathit{take}\ 10) \cdot (c\ :) \cdot \mathit{filter}(c\ \neq) \tag{3}$$

where *take* and *filter* are the obvious functions.

The calculation

You said what...? 

Software design (toy) example

The problem

Requirements fragment:

> *(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.*

The model

$$\mathit{store}\ c \quad \triangleq \quad (\mathit{take}\ 10) \cdot (c\ :) \cdot \mathit{filter}(c \neq) \tag{3}$$

where *take* and *filter* are the obvious functions.

The calculation

You said what...? 

Software design (toy) example

The solution

Following common practice, in eg. C$^{\#}$ ...

```
public void store10(string phoneNumber)
{
  System.Collections.ArrayList auxList =
      new System.Collections.ArrayList();
  auxList.Add(phoneNumber);
  auxList.AddRange(
      this.filteratmost9(phoneNumber) );
  this.callList = auxList;
}

public System.Collections.ArrayList filteratmost9(string n)
{
  System.Collections.ArrayList retList =
      new System.Collections.ArrayList();
      int i=0, m=0;
  while((i < this.callList.Count) && (m < 9))
  {
      if ((string)this.callList[i] != n)
      {
          retList.Add(this.callList[i]);
          m++;
      }
      i++;
  }
  return retList;
}
```

More than one problem

Clearly:

- **Correctness**: the **calculation** step, ie. the justification that `store10` implements the model,

$$\texttt{store10} \quad \sqsupseteq \quad store$$

  is missing.

Worse than that, the **model** itself is not yet to be trusted. Why?

- **Consistency**: the proof obligation that *store* preserves properties (1-3) of lists of calls in the mobile phone has not been discharged either.
- Example of proof obligation ignored:

$$\langle \forall \; l, c \;\; : \;\; noDuplicates \; l : \;\; noDuplicates(store \;\; c \;\; l) \rangle \qquad (4)$$

Questions

Main issue

Can we discharge **proof obligations** in program verification by **calculation**?

"First" answer:

Yes, we can use the λ-calculus, the predicate calculus etc.

"Second" answer (once you've tried it):

Yes, but that's a lot of work when tackling real-life problems. If we want to perform as **calculationally** as in other engineering disciplines, we need to bring the **algebraic structure** of the logic we are using **explicit** via some kind of transform.

What kind of **transform** do we have in mind?

Questions

Main issue

Can we discharge **proof obligations** in program verification by **calculation**?

"First" answer:

Yes, we can use the λ-calculus, the predicate calculus etc.

"Second" answer (once you've tried it):

Yes, but that's a lot of work when tackling real-life problems. If we want to perform as **calculationally** as in other engineering disciplines, we need to bring the **algebraic structure** of the logic we are using **explicit** via some kind of transform.

What kind of **transform** do we have in mind?

Questions

Main issue

Can we discharge **proof obligations** in program verification by **calculation**?

"First" answer:

Yes, we can use the λ-calculus, the predicate calculus etc.

"Second" answer (once you've tried it):

Yes, but that's a lot of work when tackling real-life problems. If we want to perform as **calculationally** as in other engineering disciplines, we need to bring the **algebraic structure** of the logic we are using **explicit** via some kind of transform.

What kind of **transform** do we have in mind?

$e = m + c$ challenges

A "notation problem":

Mathematical modelling

requires *descriptive* notations, therefore:

- intuitive
- domain-specific

Calculation

requires *elegant* notations, therefore:

- simple and compact
- generic
- cryptic, otherwise uneasy to manipulate

Recall Dijkstra's definition : *elegant* $\Leftrightarrow$ *simple and remarkably effective*

A "déjà vu" problem in engineering mathematics

Quoting Kreyszig's book, p.242: *"(...) The Laplace transformation is a method for solving differential equations (...) [which] consists of three main steps:*

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

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

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

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

Need for a transform

Integration? Quantification?

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

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

A parallel:

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

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

An "*s*-space analog" for logical quantification

The pointfree (PF) transform

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

What are R, S, id ?

Work plan

- Study the **PF-transform** and the associated **relation algebra**
- Apply the transform to proof obligations
- Discharge proof obligations by PF-calculation
- Devise a strategy for dealing with real-size software problems
- Integrate calculational style with mechanical theorem proving and model checking

We will illustrate the last item with an example taken from the **Verified File System** challenge put up by NASA JPL.

Background — Eindhoven quantifier calculus

When writing $\forall, \exists$-quantified expressions is useful to know a number of rules which help in reasoning about them. We adopt notation

$$\langle \forall\, x \;:\; R :\; T \rangle$$
$$\langle \exists\, x \;:\; R :\; T \rangle$$

meaning, respectively

- *"for all x in range R it is the case that T"*
- *"there exists x in range R such that T"*

Some useful rules about $\forall, \exists$ follow:

- **Trading**:

$$\langle \forall\, i \;:\; R \wedge S :\; T \rangle \;=\; \langle \forall\, i \;:\; R :\; S \Rightarrow T \rangle \tag{5}$$
$$\langle \exists\, i \;:\; R \wedge S :\; T \rangle \;=\; \langle \exists\, i \;:\; R :\; S \wedge T \rangle \tag{6}$$

Background — Eindhoven quantifier calculus

One-point:

$$\langle \forall\ k\ :\ k = e :\ T \rangle \quad = \quad T[k := e] \tag{7}$$
$$\langle \exists\ k\ :\ k = e :\ T \rangle \quad = \quad T[k := e] \tag{8}$$

de Morgan:

$$\neg\langle \forall\ i\ :\ R :\ T \rangle \quad = \quad \langle \exists\ i\ :\ R :\ \neg T \rangle \tag{9}$$
$$\neg\langle \exists\ i\ :\ R :\ T \rangle \quad = \quad \langle \forall\ i\ :\ R :\ \neg T \rangle \tag{10}$$

Nesting:

$$\langle \forall\ a, b\ :\ R \wedge S :\ T \rangle \quad = \quad \langle \forall\ a\ :\ R :\ \langle \forall\ b\ :\ S :\ T \rangle\rangle \tag{11}$$
$$\langle \exists\ a, b\ :\ R \wedge S :\ T \rangle \quad = \quad \langle \exists\ a\ :\ R :\ \langle \exists\ b\ :\ S :\ T \rangle\rangle \tag{12}$$

Background — Eindhoven quantifier calculus

Empty range:

$$\langle \forall\ k\ :\ \textsc{False} :\ T \rangle \quad = \quad \textsc{True} \tag{13}$$

$$\langle \exists\ k\ :\ \textsc{False} :\ T \rangle \quad = \quad \textsc{False} \tag{14}$$

Splitting:

$$\langle \forall\ j\ :\ R :\ \langle \forall\ k\ :\ S :\ T \rangle\rangle \quad = \quad \langle \forall\ k\ :\ \langle \exists\ j\ :\ R :\ S \rangle :\ T \rangle \tag{15}$$

$$\langle \exists\ j\ :\ R :\ \langle \exists\ k\ :\ S :\ T \rangle\rangle \quad = \quad \langle \exists\ k\ :\ \langle \exists\ j\ :\ R :\ S \rangle :\ T \rangle \tag{16}$$

etc. [3]

Exercises (warming up)

Exercise 1: Show that equivalences (1) and (2) hold.

□

Exercise 2: Consider the following variant of the *al-hatt* rule restricted to (positive) natural numbers:

$$x * z \leq y \quad \Leftrightarrow \quad x \leq y / z \tag{17}$$

where y/z denotes the *integral division* of y by z, eg. such that $3/2 = 1$, etc. Resort directly to (17) in showing that $y/0$ is the largest of all natural numbers.

□

Second lecture

Schedule: Monday Feb 25th, 17h20-18h10

Learning outcomes:

- **PF-transform** essentials
- Binary relation **combinators**. The role of composition.
- Taxonomy of binary relations. **Functions**. "Al-djabr" rules.
- Rules of the PF-transform.

Pairs

Consider assertions

$$\begin{array}{rcl} \texttt{John} & \mathit{IsFatherOf} & \texttt{Mary} \\ 3 & =(1+) & 2 \\ P & \sqsupseteq & S \end{array}$$

- They are statements of fact concerning various kinds of object — people, natural numbers, programs and specifications, etc
- They involve *two* such objects, that is, **pairs**

$$(\texttt{John}, \texttt{Mary})$$
$$(3, 2)$$
$$(P, S)$$

respectively.

Sets of pairs

So, we might have written

$$\begin{array}{rcl} (\texttt{John}, \texttt{Mary}) & \in & \mathit{IsFatherOf} \\ (3, 2) & \in & (1+) \\ (P, S) & \in & \sqsupseteq \end{array}$$

What are $\mathit{IsFatherOf}$, $(1+)$, $(\sqsupseteq)$?

- they are **sets of pairs**
- they are **binary relations**

Therefore,

- **functions** — eg. $\mathit{succ} \triangleq (1+)$ — are special cases of relations

as well as partial **orders** — eg. $(\leq)$, etc

Binary Relations

Binary relations are typed:

Arrow notation

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

A, B are types. Writing $B \xleftarrow{R} A$ means the same as $A \xrightarrow{R} B$.

Infix notation

The usual infix notation used in natural language — eg. `John` *IsFatherOf* `Mary` — and in maths — eg. $0 \leq \pi$ — extends to arbitrary $B \xleftarrow{R} A$: we write

$$b \; R \; a$$

to denote that $(b, a) \in R$.

Binary Relations

Binary relations are typed:

Arrow notation

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

A, B are types. Writing $B \xleftarrow{R} A$ means the same as $A \xrightarrow{R} B$.

Infix notation

The usual infix notation used in natural language — eg. $\texttt{John}$ *IsFatherOf* $\texttt{Mary}$ — and in maths — eg. $0 \leq \pi$ — extends to arbitrary $B \xleftarrow{R} A$: we write

$$b \; R \; a$$

to denote that $(b, a) \in R$.

Functions are relations

- Lowercase letters (or identifiers starting by one such letter) will denote special relations known as **functions**, eg. f, g, $succ$, etc.
- We regard **function** $f : A \longrightarrow B$ as the binary **relation** which relates b to a iff $b = f\ a$. So,

$$b\ f\ a \text{ literally means } b = f\ a$$

- Therefore, we generalize

$$\boxed{\begin{array}{c} B \xleftarrow{f} A \\ b = f\ a \end{array}} \quad \text{to} \quad \boxed{\begin{array}{c} B \xleftarrow{R} A \\ b\ R\ a \end{array}}$$

Composition

Recall **function composition**

$$\begin{tikzcd} B & A \arrow[l, "f"'] & C \arrow[l, "g"'] \arrow[ll, bend left, "f \cdot g"] \end{tikzcd} \tag{18}$$

$$b = f(g\ c)$$

and extend $f \cdot g$ to $R \cdot S$ in the obvious way:

$$b(R \cdot S)c \quad \Leftrightarrow \quad \langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle \tag{19}$$

Note how this rule of the PF-transform *removes* $\exists$ when applied from right to left.

Check generalization

Back to functions, (19) becomes

$$
\begin{array}{cl}
b(f \cdot g)c & \Leftrightarrow \quad \langle \exists\ a\ ::\ b\ f\ a \wedge a\ g\ c \rangle \\
& \Leftrightarrow \qquad \{ \ a\ g\ c \text{ means } a = g\ c \ \} \\
& \quad\ \ \langle \exists\ a\ ::\ b\ f\ a \wedge a = g\ c \rangle \\
& \Leftrightarrow \qquad \{ \ \exists\text{-trading} \ ; \ b\ f\ a \text{ means } b = f\ a \ \} \\
& \quad\ \ \langle \exists\ a\ :\ a = g\ c\ :\ b = f\ a \rangle \\
& \Leftrightarrow \qquad \{ \ \text{one-point rule } (\exists) \ \} \\
& \quad\ \ b = f(g\ c)
\end{array}
$$

So, we easily recover what we had before (18).

Inclusion generalizes equality

- **Equality** on functions $B \xleftarrow{f,g} A$

$$f = g \;\Leftrightarrow\; \langle \forall\, a \;:\; a \in A :\; f\ a \;=_B\; g\ a \rangle$$

  generalizes to **inclusion** on relations:

$$R \subseteq S \;\Leftrightarrow\; \langle \forall\, b, a \;::\; b\ R\ a \Rightarrow b\ S\ a \rangle \tag{20}$$

  (read $R \subseteq S$ as "R is at most S")
- $R \subseteq S$ is a partial order — reflexive, transitive and anti-symmetric
- **Equality** on relations $B \xleftarrow{R,S} A$:

$$R = S \;\Leftrightarrow\; R \subseteq S \wedge S \subseteq R \tag{21}$$

Special relations

Every type $B \longleftarrow A$ has its

- *bottom* relation $B \xleftarrow{\perp} A$, which is such that, for all b, a, $b \perp a \;\Leftrightarrow\; \textsc{False}$
- *topmost* relation $B \xleftarrow{\top} A$, which is such that, for all b, a, $b \top a \;\Leftrightarrow\; \textsc{True}$

Type $A \longleftarrow A$ has the

- identity relation $A \xleftarrow{id} A$ which is function $id\ a \triangleq a$.

Clearly, for every R,

$$\perp \;\subseteq\; R \;\subseteq\; \top \tag{22}$$

Exercises

Exercise 3: Resort to PF-transform rule (19) and to the Eindhoven quantifier calculus to show that

$$R \cdot id = R = id \cdot R \tag{23}$$

$$R \cdot \bot = \bot = \bot \cdot R \tag{24}$$

hold and that composition is associative:

$$R \cdot (S \cdot T) = (R \cdot S) \cdot T \tag{25}$$

□

Converses

Every relation $B \xleftarrow{R} A$ has a **converse** $B \xrightarrow{R^\circ} A$ which is such that, for all a, b,

$$a(R^\circ)b \quad \Leftrightarrow \quad b\ R\ a \tag{26}$$

Note that converse commutes with composition

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{27}$$

and cancels itself

$$(R^\circ)^\circ = R \tag{28}$$

— two corollaries of "al-djabr" rule

$$R^\circ \subseteq S \ \Leftrightarrow \ R \subseteq S^\circ \tag{29}$$

Function converses

Function converses f°, g° etc. always exist (as **relations**) and enjoy the following (very useful) PF-transform property:

$$(f\ b)R(g\ a) \quad\Leftrightarrow\quad b(f^\circ \cdot R \cdot g)a \tag{30}$$

cf. diagram:

$$\begin{tikzcd}
B \arrow[d, "f^\circ"'] & A \arrow[l, "R"'] \\
C & D \arrow[l, "f^\circ \cdot R \cdot g"] \arrow[u, "g"']
\end{tikzcd}$$

Let us see an example of its use.

PF-transform at work

Transforming a well-known PW-formula:

$$
\begin{array}{ll}
& f \text{ is injective} \\
\Leftrightarrow & \qquad \{ \text{ recall definition from discrete maths } \} \\
& \langle \forall\, y, x \;::\; (f\ y) = (f\ x) \Rightarrow y = x \rangle \\
\Leftrightarrow & \qquad \{ \text{ introduce } id \text{ (twice) } \} \\
& \langle \forall\, y, x \;::\; (f\ y) id (f\ x) \Rightarrow y(id)x \rangle \\
\Leftrightarrow & \qquad \{ \text{ rule } (f\ b)R(g\ a) \Leftrightarrow b(f^\circ \cdot R \cdot g)a \text{ (30) } \} \\
& \langle \forall\, y, x \;::\; y(f^\circ \cdot id \cdot f)x \Rightarrow y(id)x \rangle \\
\Leftrightarrow & \qquad \{ \text{ (23) ; then go pointfree via (20) } \} \\
& f^\circ \cdot f \subseteq id
\end{array}
$$

The other way round

Let us now see what $id \subseteq f \cdot f^\circ$ means:

$$
\begin{array}{ll}
& id \subseteq f \cdot f^\circ \\
\Leftrightarrow & \quad \{ \text{ relational inclusion (20) } \} \\
& \langle \forall\, y, x \;::\; y(id)x \Rightarrow y(f \cdot f^\circ)x \rangle \\
\Leftrightarrow & \quad \{ \text{ identity relation ; composition (19) } \} \\
& \langle \forall\, y, x \;::\; y = x \Rightarrow \langle \exists\, z \;::\; y\ f\ z \wedge z\ f^\circ x \rangle \rangle \\
\Leftrightarrow & \quad \{ \text{ } \forall\text{-trading ; converse (26) } \} \\
& \langle \forall\, y, x \;:\; y = x : \langle \exists\, z \;::\; y\ f\ z \wedge x\ f\ z \rangle \rangle \\
\Leftrightarrow & \quad \{ \text{ } \forall\text{-one point ; trivia ; function } f \text{ } \} \\
& \langle \forall\, x \;::\; \langle \exists\, z \;::\; x = f\ z \rangle \rangle \\
\Leftrightarrow & \quad \{ \text{ recal definition from maths } \} \\
& f \text{ is surjective}
\end{array}
$$

Why *id* (really) matters

Terminology:

- Say R *is reflexive* iff $id \subseteq R$
  pointwise: $\langle \forall\ a\ ::\ a\ R\ a \rangle$ (check as homework);
- Say R *is coreflexive* iff $R \subseteq id$
  pointwise: $\langle \forall\ a\ ::\ b\ R\ a \Rightarrow b = a \rangle$ (check as homework).

Define, for $B \xleftarrow{R} A$:

Kernel of R	**Image** of R
$A \xleftarrow{\ker R} A$ $\ker R \triangleq R^\circ \cdot R$	$B \xleftarrow{\mathrm{img}\, R} B$ $\mathrm{img}\, R \triangleq R \cdot R^\circ$

Example: kernels of functions

$$
\begin{array}{cl}
 & a'(\ker f)a \\
\Leftrightarrow & \{ \text{ substitution } \} \\
 & a'(f^\circ \cdot f)a \\
\Leftrightarrow & \{ \text{ PF-transform rule (30) } \} \\
 & (f\ a') = (f\ a)
\end{array}
$$

In words: $a'(\ker f)a$ means a' *and* a *"have the same* f*-image"*

Exercise 4: Let C be a nonempty data domain and let and $c \in C$. Let $\underline{c}$ be the *"everywhere* c*"* function:

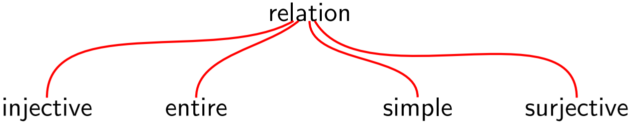
$$
\begin{array}{rcl}
\underline{c} & : & A \longrightarrow C \\
\underline{c}\, a & \triangleq & c
\end{array}
\qquad (31)
$$

Compute which relations are defined by the following PF-expressions:

$$
\ker \underline{c} \quad , \quad \underline{b} \cdot \underline{c}^\circ \quad , \quad \mathrm{img}\, \underline{c} \qquad (32)
$$

Binary relation taxonomy

Topmost criteria:



Definitions:

	Reflexive	*Coreflexive*
$\ker R$	entire R	injective R
$\mathrm{img}\, R$	surjective R	simple R

(33)

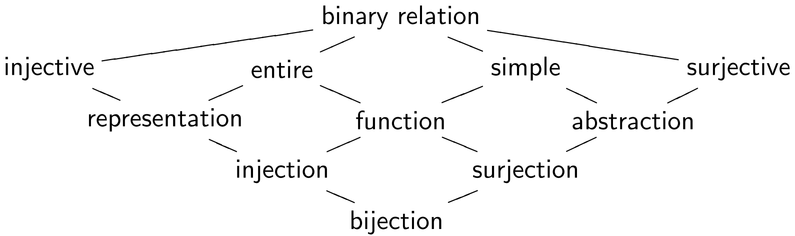
Facts:

$$\ker (R^\circ) = \mathrm{img}\, R \tag{34}$$

$$\mathrm{img}\, (R^\circ) = \ker R \tag{35}$$

Binary relation taxonomy

The whole picture:



(36)

Exercise 5: Resort to (34,35) and (33) to prove the following rules of thumb:

- converse of **injective** is **simple** (and vice-versa)
- converse of **entire** is **surjective** (and vice-versa)

□

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)

which both together are equivalent to any of "al-djabr" rules

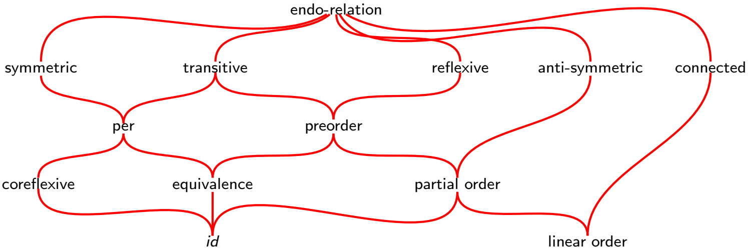
$$f \cdot R \subseteq S \;\Leftrightarrow\; R \subseteq f^{\circ} \cdot S \tag{37}$$

$$R \cdot f^{\circ} \subseteq S \;\Leftrightarrow\; R \subseteq S \cdot f \tag{38}$$

Notation convention: functions will be denoted by lowercase characters (eg. f, g, ϕ) or identifiers starting with lowercase characters, and function application will be denoted by juxtaposition, eg. $f\ a$ instead of $f(a)$.

Relation taxonomy — orders

Orders are endo-relations $A \xleftarrow{R} A$ classified as



(Criteria definitions: next slide)

Orders and their taxonomy

Besides

reflexive:	iff $id_A \subseteq R$
coreflexive:	iff $R \subseteq id_A$

an order (or endo-relation) $A \xleftarrow{R} A$ can be

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

Orders and their taxonomy

Therefore:

- **Preorders** are reflexive and transitive orders.
  Example: $y\ \mathit{IsAtMostAsOldAs}\ x$
- **Partial** orders are anti-symmetric preorders
  Example: $y \subseteq x$
- **Linear** orders are connected partial orders
  Example: $y \leq x$
- **Equivalences** are symmetric preorders
  Example: $y\ \mathit{Permutes}\ x$ (lists)
- **Pers** are partial equivalences
  Example: $y\ \mathit{IsBrotherOf}\ x$

Exercises

Exercise 6: Expand all criteria in the previous slides to pointwise notation.

□

Exercise 7: A relation R is said to be *co-transitive* iff the following holds:

$$\langle \forall\ b, a\ :\ b\ R\ a : \langle \exists\ c\ :\ b\ R\ c :\ c\ R\ a \rangle\rangle \tag{39}$$

Compute the PF-transform of the formula above. Find a relation (eg. over numbers) which is co-transitive and another which is not.

□

Meet and join

Meet (intersection) and join (union) internalize conjunction and disjunction, respectively,

$$b\ (R \cap S)\ a \quad \Leftrightarrow \quad b\ R\ a \wedge b\ S\ a \tag{40}$$

$$b\ (R \cup S)\ a \quad \Leftrightarrow \quad b\ R\ a \vee b\ S\ a \tag{41}$$

for R, S of the same type. Their meaning is captured by the following **universal** properties:

$$X \subseteq R \cap S \quad \Leftrightarrow \quad X \subseteq R \ \wedge\ X \subseteq S \tag{42}$$

$$R \cup S \subseteq X \quad \Leftrightarrow \quad R \subseteq X \ \wedge\ S \subseteq X \tag{43}$$

NB: these are also “al-djabr” rules, although slightly more elaborate than those seen so far.

In summary

Type $B \longleftarrow A$ forms a lattice:

$$
\begin{tikzcd}[column sep=small]
 & \top \arrow[d, dash, dotted] & & & \text{“top”} \\
 & R \cup S \arrow[dl, dash, dotted] \arrow[dr, dash, dotted] & & & \text{join, lub (“least upper bound”)} \\
R \arrow[dr, dash] & & S \arrow[dl, dash] & & \\
 & R \cap S \arrow[d, dash, dotted] & & & \text{meet, glb (“greatest lower bound”)} \\
 & \bot & & & \text{“bottom”}
\end{tikzcd}
$$

All (data structures) **in one** (PF notation)

Products

$$
\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
 & C \arrow[ul, "R"] \arrow[u, "\langle R,S \rangle"'] \arrow[ur, "S"'] &
\end{tikzcd}
\tag{44}
$$

where

$$
\begin{array}{c|c}
\psi & PF\ \psi \\
\hline\hline
a\ R\ c \wedge b\ S\ c & (a,b)\langle R,S\rangle c \\
b\ R\ a \wedge d\ S\ c & (b,d)(R \times S)(a,c)
\end{array}
\tag{45}
$$

Clearly: $R \times S = \langle R \cdot \pi_1, S \cdot \pi_2 \rangle$

Sums

Example (Haskell)

```
data X = Boo Bool | Err String
```

PF-transforms to:

$$
\begin{tikzcd}
Bool \arrow[r, "i_1"] \arrow[dr, "Boo"'] & Bool + String \arrow[d, "{[Boo\ , Err]}"] & String \arrow[l, "i_2"'] \arrow[dl, "Err"] \\
& X &
\end{tikzcd}
\tag{46}
$$

where

$$[R\ , S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \quad \text{cf.} \quad
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "R"'] & A + B \arrow[d, "{[R\ , S]}"] & B \arrow[l, "i_2"'] \arrow[dl, "S"] \\
& C &
\end{tikzcd}$$

Dually: $R + S = [i_1 \cdot R\ , i_2 \cdot S]$

Sums

Example (Haskell)

```
data X = Boo Bool | Err String
```

PF-transforms to:

$$
\begin{tikzcd}
Bool \arrow[r, "i_1"] \arrow[dr, "Boo"'] & Bool + String \arrow[d, "{[Boo\ ,Err]}"] & String \arrow[l, "i_2"'] \arrow[dl, "Err"] \\
& X &
\end{tikzcd}
\tag{46}
$$

where

$$[R\ ,S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)$$ cf. $$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "R"'] & A + B \arrow[d, "{[R\ ,S]}"] & B \arrow[l, "i_2"'] \arrow[dl, "S"] \\
& C &
\end{tikzcd}$$

Dually: $R + S = [i_1 \cdot R\ ,i_2 \cdot S]$

Useful consequence of "al-djabr" rules

- All relational combinators involved in "al-djabr" rules are **monotonic**
- The ones on the lower side of rules distribute over $\cup$, eg.:

$$(R \cup S)^\circ = R^\circ \cup S^\circ \tag{47}$$

$$f \cdot (R \cup S) = f \cdot R \cup f \cdot S \tag{48}$$

- The ones on the upper side of rules distribute over $\cap$, eg.:

$$(R \cap S)^\circ = R^\circ \cap S^\circ \tag{49}$$

$$(R \cap S) \cdot f = R \cdot f \cap S \cdot f \tag{50}$$

Exercises

Exercise 8: Prove the following rules of thumb:

- *smaller than injective (simple) is injective (simple)*
- *larger than entire (surjective) is entire (surjective)*

□

Exercise 9: Check which of the following hold:

- If relations R and S are simple, then so is $R \cap S$
- If relations R and S are injective, then so is $R \cup S$
- If relations R and S are entire, then so is $R \cap S$

□

Exercises

Exercise 10: Prove that relational composition preserves *all* relational classes in the taxonomy of (36).

□

Exercise 11: Show that the PW definition of $\langle R, S \rangle$ given above PF-transforms to

$$\langle R, S \rangle \quad = \quad \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \tag{51}$$

□

Exercise 12: Infer "al-djabr" rule

$$X \subseteq \langle R, S \rangle \quad \Leftrightarrow \quad \pi_1 \cdot X \subseteq R \wedge \pi_2 \cdot X \subseteq S \tag{52}$$

from (51) and (42).

□

Exercises

Exercise 13: Prove the following fact

$$\textit{A function } f \textit{ is a bijection iff its converse } f^{\circ} \textit{ is a function} \tag{53}$$

by completing:

$$
\begin{array}{ll}
 & f \text{ and } f^{\circ} \text{ are functions} \\
\Leftrightarrow & \quad \{ \ldots \} \\
 & (id \subseteq \ker f \wedge \operatorname{img} f \subseteq id) \wedge (id \subseteq \ker f^{\circ} \wedge \operatorname{img} f^{\circ} \subseteq id) \\
\Leftrightarrow & \quad \{ \ldots \} \\
 & \vdots \\
\Leftrightarrow & \quad \{ \ldots \} \\
 & f \text{ is a bijection}
\end{array}
$$

$\square$

Exercises

Exercise 14: Prove that $swap \triangleq \langle \pi_2, \pi_1 \rangle$ is a bijection.

□

Exercise 15: Show that (30) holds.

□

Notation: simple relations will be singled out in diagrams by drawing $A \rightharpoonup B$ instead of $B \rightarrow A$. Arrows labelled with lowercase letters denote functions.

Summary

Rules of the PF-transform seen so far:

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

Using the Alloy Analyser as a PF-transform checker

- **Alloy** model checker (`http://alloy.mit.edu`) — simple and elegant
- In Alloy, *"everything is a relation"*
- Example of pointwise Alloy:

```
pred Injective {
all x, y : A, z : B | z in x.R && z in y.R => x=y
}
```

NB: note the transposed notation `x.R` meaning set $\{y : y\ R\ x\}$.

Using the Alloy Analyser as a PF-transform checker

The same in pointfree Alloy:

```
pred Injective' {
R.~R in iden :> A
}
```

— recall $R^\circ \cdot R \subseteq id$. Alternatively, we may write

```
pred Injective'' {
R in A lone -> B
}
pred Injective''' {
all x : B | lone R.x
}
```

Using the Alloy Analyser as a PF-transform checker

The checking process itself: run eg.

```
check { Simple <=> Injective }
```

where

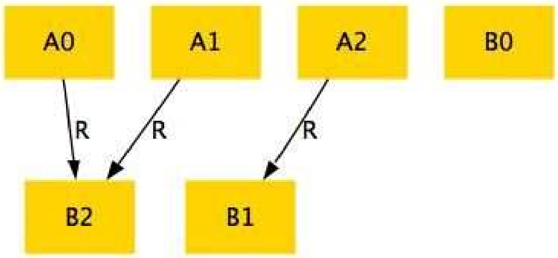
```
pred Simple {
~R.R in iden:> B
}
```

Alloy's answer:

Executing "Check assert$2"
Solver=sat4j Bitwidth=4 MaxSeq=4 Symmetry=20
124 vars. 15 primary vars. 280 clauses. 55ms.
Counterexample found. Assertion is invalid. 105ms.

Using the Alloy Analyser as a PF-transform checker

Alloy counter example as shown by the tool:



To see the nice blend between Alloy and the PF-transform have a look at `RelCalc.als` in `src_alloy.tar.bz`, available from `http://twiki.di.uminho.pt/twiki/bin/view/Research/VFS/WebHome`.

Third lecture

Schedule: Tuesday Feb 26th, 16h20-17h10

Learning outcomes:

- Data-type invariants. PF-transform of unary predicates. Coreflexives and conditions.
- Proof obligations (PO): invariant preservation.
- PF-transformed POs.
- Relationship with Hoare logic.

Types for software quality

Data type evolution:

- **Assembly** (1950s) — one single primitive data type: machine binary
- **Fortran** (1960s) — primitive types for numeric processing (INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL data types)
- **Pascal** (1970s) — user defined (**monomorphic**) data types (eg. records, files)
- **ML**, **Haskell** etc ($\geq$1980s) — user defined (**polymorphic**) data types (eg. $List\ a$ for all a)

Type checking for software quality

Why data types?

- **Fortran** anecdote: non-terminating loop `DO I = 1.10` once went unnoticed due to poor type-checking
- Diagnosis: compiler unable to prevent using a real number where a discrete value (eg. integer, enumerated type) was expected
- Solution: improve grammar + static type checker

(static means *done at compile time*)

Data type invariants

In a system for monitoring the flight paths of aircrafts in a controlled airspace, we need to define altitude, latitude and longitude:

$$\begin{aligned} Alt &= \mathbb{R} \\ Lat &= \mathbb{R} \\ Lon &= \mathbb{R} \end{aligned}$$

However,

- altitude cannot be negative
- latitude ranges between -90 and 90
- longitude ranges between -180 and 180

In maths we would have defined:

$$\begin{aligned} Alt &= \{a \in \mathbb{R} : a \geq 0\} \\ Lat &= \{x \in \mathbb{R} : -90 \leq x \leq 90\} \\ Lon &= \{y \in \mathbb{R} : -180 \leq y \leq 180\} \end{aligned}$$

Data type invariants

In a system for monitoring the flight paths of aircrafts in a controlled airspace, we need to define altitude, latitude and longitude:

$$\begin{aligned} Alt &= \mathbb{R} \\ Lat &= \mathbb{R} \\ Lon &= \mathbb{R} \end{aligned}$$

However,

- altitude cannot be negative
- latitude ranges between -90 and 90
- longitude ranges between -180 and 180

In maths we would have defined:

$$\begin{aligned} Alt &= \{a \in \mathbb{R} : a \geq 0\} \\ Lat &= \{x \in \mathbb{R} : -90 \leq x \leq 90\} \\ Lon &= \{y \in \mathbb{R} : -180 \leq y \leq 180\} \end{aligned}$$

Data type invariants "a la" VDM

Standard notation (VDM family)

$$\begin{array}{l} Alt \;=\; \mathbb{R} \\ \textbf{inv}\ a \triangleq\ a \geq 0 \end{array}$$

implicitly defines predicate

$$\begin{array}{l} inv\text{-}Alt : \mathbb{R} \to \mathbb{B} \\ inv\text{-}Alt(a) \triangleq\ a \geq 0 \end{array}$$

known as the *invariant* of Alt.

Invariants are *inevitable*

Modeling the Western dating system:

$$Year = \mathbb{N}$$

$$Month = \mathbb{N}$$
$$\textbf{inv } m \triangleq \ m \leq 12$$

$$Day = \mathbb{N}$$
$$\textbf{inv } d \triangleq \ d \leq 31$$

$$Date = Year \times Month \times Day$$

However, $12 \times 31 = 372$, while one year has 365.2425... days. Thus the need for *leap years* in the *Julian calendar* (45 BC) and in the *Gregorian calendar* (1582), leading to

Invariants are *inevitable*

$$
\begin{array}{ll}
\mathit{Date} = \mathit{Year} \times \mathit{Month} \times \mathit{Day} & \\
\mathbf{inv}(y, m, d) \triangleq & \mathit{if}\ \ m \in \{1, 3, 5, 7, 8, 10, 12\}\ \ \mathit{then} \\
& d \leq 31 \wedge \\
& ((y = 1582 \wedge m = 10) \Rightarrow (d < 5 \vee 14 < d)) \\
& \mathit{else\ if}\ \ m \in \{4, 6, 9, 11\}\ \ \mathit{then}\ d \leq 30 \\
& \mathit{else\ if}\ \ m = 2 \wedge \mathit{leapYear}(y)\ \ \mathit{then}\ d \leq 29 \\
& \mathit{else\ if}\ \ m = 2 \wedge \neg \mathit{leapYear}(y)\ \ \mathit{then}\ d \leq 28 \\
& \mathit{else}\ \textsc{False};
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathit{leapYear} & : & \mathbb{N} \rightarrow \mathbb{B} \\
\mathit{leapYear}\ y & \triangleq & 0 = \mathit{rem}(y, \mathit{if}\ \ y \geq 1700 \wedge \mathit{rem}(y, 100) = 0 \\
& & \mathit{then}\ 400\ \ \mathit{else}\ \ 4)
\end{array}
$$

Summing up

- Given a datatype A and a predicate $p : A \to \mathbb{B}$, data type declaration

$$\begin{array}{lcl} T & = & A \\ \mathbf{inv}\ x & \triangleq & p\ x \end{array}$$

  means the type whose extension is

$$T = \{x \in A : p\ x\}$$

- p is referred to as the invariant property of T
- Therefore, writing $a \in T$ means $a \in A \wedge (p\ a)$.
- A itself can have an invariant, so the process is inductive on the structure of types.

Invariants entail proof obligations

Consider

$$\begin{array}{l} \mathit{Even} \;=\; \mathbb{N} \\ \textbf{inv}\ x \triangleq\ \mathit{even}\ x \end{array}$$

where $\mathit{even}\ n \;\triangleq\; \langle \exists\, k \;::\; n = 2k \rangle$ and

$$\begin{array}{l} \mathit{twice} : \mathit{Even} \rightarrow \mathit{Even} \\ \mathit{twice}\ n \triangleq\ 2n \end{array}$$

Proof obligation

$$\langle \forall\, x, y \;:\; \mathit{even}\ x \wedge y = \mathit{twice}\ x :\; \mathit{even}\ y \rangle \tag{54}$$

expresses the fact that function *twice* preserves even numbers.

Invariants entail proof obligations

Given proposed **model** for operation *store* in the mobile phone problem,

$$store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls$$
$$store\ c\ l \triangleq\ take\ 10\ (c : [\ a \mid a \leftarrow l, a \neq c\ ])$$

the fact that *ListOfCalls* has **invariant**

$$ListOfCalls = Call^{\star}$$
$$\textbf{inv}\ \ l \triangleq\ length\ l \leq 10\ \wedge$$
$$\langle \forall\ i, j\ :\ 1 \leq i, j \leq length\ l :\ (l\ i) = (l\ j) \Rightarrow i = j \rangle$$

leads to **proof obligation**

$$\langle \forall\ c, l\ :\ l \in ListOfCalls :\ store\ c\ l \in ListOfCalls \rangle \qquad (55)$$

Dealing with proof obligations

- In full-fledged formal techniques, one is obliged to provide a **mathematical proof** that conjectures such as (56) do hold for **any** a.
- Such proofs can either be performed as paper-and-pencil exercises or, in case of very complex invariants, be supported by **theorem provers**
- If automatic, discharging such proofs can be regarded as **extended static checking** (ESC)
- As we shall see, *all* the above approaches to adding quality to a formal model are useful and have their place in software engineering using formal methods.

PF-ESC

- The main novelty of our approach resides in the chosen method of proof construction: first-order proof obligations are subject to the **PF-transform** before they are reasoned about.
- This transformation eliminates quantifiers and bound variables and reduces complex formulas to algebraic relational expressions which are more **agile** to calculate with.
- Suitable relational encoding of recursive structures makes it possible to perform **non-inductive** proofs over such structures.

However

- The PF-transform seems applicable to transforming **binary** predicates only, easily converted to binary relations, eg. $\phi(y,x) \triangleq\ y-1=2x$ which transforms to function $y=2x+1$, etc.
- What about transforming predicates such as *even* in (54)?
- As already noted, (54) is a proposition stating that function *twice* *preserves* even numbers.
- In general, a function $A \xleftarrow{f} A$ is said to **preserve** a given predicate ϕ iff the following holds:

$$\langle \forall\ x\ :\ \phi\ x :\ \phi\ (f\ x) \rangle \tag{56}$$

Answer

First PF-transform scope of (54):

$$
\begin{array}{ll}
& y = \mathit{twice}\ x \wedge \mathit{even}\ x \\
\Leftrightarrow & \qquad \{\ \exists\text{-one-point}\ \} \\
& \langle \exists\ z\ :\ z = x :\ y = \mathit{twice}\ z \wedge \mathit{even}\ z \rangle \\
\Leftrightarrow & \qquad \{\ \exists\text{-trading}\ ;\ \text{introduce}\ \Phi_{\mathit{even}}\ \} \\
& \langle \exists\ z\ ::\ y = \mathit{twice}\ z \wedge \underbrace{z = x \wedge \mathit{even}\ z}_{z\ \Phi_{\mathit{even}}\ x} \rangle \\
\Leftrightarrow & \qquad \{\ \text{composition}\ \} \\
& y(\mathit{twice} \cdot \Phi_{\mathit{even}})x
\end{array}
$$

cf. diagram

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "\mathit{twice}"'] & \mathbb{N}_0 \arrow[l, "\Phi_{\mathit{even}}"'] \\
\mathbb{N}_0 &
\end{tikzcd}
$$

Now the whole thing

$$\langle \forall\, x, y \;:\; y = twice\ x \wedge even\ x :\; even\ y \rangle$$

$\Leftrightarrow$ { above }

$$\langle \forall\, x, y \;:\; y(twice \cdot \Phi_{even})x :\; even\ y \rangle$$

$\Leftrightarrow$ { $\exists$-one-point }

$$\langle \forall\, x, y \;:\; y(twice \cdot \Phi_{even})x :\; \langle \exists\, z \;:\; z = y :\; even\ z \rangle \rangle$$

$\Leftrightarrow$ { predicate calculus: $p \wedge \textsc{True} = p$ }

$$\langle \forall\, x, y \;:\; y(twice \cdot \Phi_{even})x :\; \langle \exists\, z \;::\; y = x \wedge even\ z \wedge \textsc{True} \rangle \rangle$$

$\Leftrightarrow$ { $\top$ is the topmost relation }

$$\langle \forall\, x, y \;:\; y(twice \cdot \Phi_{even})x :\; \langle \exists\, z \;::\; y\ \Phi_{even}\ z \wedge z \top x \rangle \rangle$$

$\Leftrightarrow$ { composition ; trading (5) }

Now the whole thing

$$\begin{array}{ll} & \langle \forall\, x, y \;::\; y(twice \cdot \Phi_{even})x \Rightarrow y(\Phi_{even} \cdot \top)x \rangle \\ \Leftrightarrow & \qquad \{ \text{ go pointfree (inclusion) } \} \\ & twice \cdot \Phi_{even} \;\subseteq\; \Phi_{even} \cdot \top \end{array} \tag{57}$$

cf. diagram

$$\begin{tikzcd} \mathbb{N}_0 \arrow[d, "twice"'] & \mathbb{N}_0 \arrow[l, "\Phi_{even}"'] \arrow[d, "\top"] \arrow[dl, phantom, "\subseteq"] \\ \mathbb{N}_0 & \mathbb{N}_0 \arrow[l, "\Phi_{even}"] \end{tikzcd}$$

In summary

In the calculation above, **unary** predicate *even* has been PF-transformed in two ways:

- Φ_{even} such that

$$z\ \Phi_{even}\ x \ \triangleq\ z = x \wedge even\ z$$

  — that is, Φ_{even} is a **coreflexive** relation;

- $\Phi_{even} \cdot \top$, which is such that

$$z(\Phi_{even} \cdot \top)x \ \Leftrightarrow\ even\ z$$

  — a so-called (left) *condition*.

Coreflexives

The PF-transformation of **unary** predicates to fragments of *id* coreflexives) is captured by the following universal property:

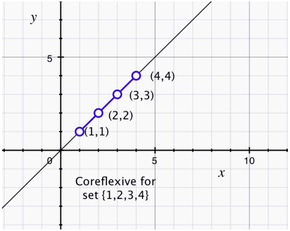
$$\Psi = \Phi_p \;\Leftrightarrow\; (y \; \Psi \; x \Leftrightarrow y = x \wedge p \; y) \tag{58}$$

Via cancellation, (58) yields

$$y \; \Phi_p \; x \;\Leftrightarrow\; y = x \wedge p \; y \tag{59}$$

A set S can also be PF-transformed into a coreflexive by calculating $\Phi_{(\in S)}$, cf. eg. the transform of set $\{1,2,3,4\}$:

$$\Phi_{1 \leq x \leq 4} \quad = $$



Boolean algebra of coreflexives

Building up one the exercises above, from (58) one easily draws:

$$\Phi_{p \wedge q} = \Phi_p \cdot \Phi_q \tag{60}$$
$$\Phi_{p \vee q} = \Phi_p \cup \Phi_q \tag{61}$$
$$\Phi_{\neg p} = id - \Phi_p \tag{62}$$
$$\Phi_{false} = \bot \tag{63}$$
$$\Phi_{true} = id \tag{64}$$

where p, q are predicates.

(Note the slight, obvious abuse in notation.)

Basic properties of coreflexives

Let Φ, Ψ be coreflexive relations. Then the following properties hold:

- Coreflexives are **symmetric** and **transitive**:

$$\Phi^\circ = \Phi = \Phi \cdot \Phi \tag{65}$$

- **Meet** of two coreflexives is composition:

$$\Phi \cap \Psi = \Phi \cdot \Psi \tag{66}$$

- **Pre** and **post** restriction:

$$R \cdot \Phi = R \cap \top \cdot \Phi \tag{67}$$

$$\Psi \cdot R = R \cap \Psi \cdot \top \tag{68}$$

Back to the twice/even example

We are now in position to get rid of $\top$ in (57):

$$\begin{array}{cl}
& twice \cdot \Phi_{even} \subseteq \Phi_{even} \cdot \top \\
\Leftrightarrow & \{ \ \Phi_{even} \subseteq id \ \} \\
& twice \cdot \Phi_{even} \subseteq \Phi_{even} \cdot \top \wedge twice \cdot \Phi_{even} \subseteq twice \\
\Leftrightarrow & \{ \ \cap\text{-universal } (42) \ \} \\
& twice \cdot \Phi_{even} \subseteq \Phi_{even} \cdot \top \cap twice \\
\Leftrightarrow & \{ \ \text{post restriction rule } (68) \ \} \\
& twice \cdot \Phi_{even} \subseteq \Phi_{even} \cdot twice
\end{array}$$

cf. diagram

$$\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "twice"'] & \mathbb{N}_0 \arrow[l, "\Phi_{even}"'] \arrow[d, "twice"] \arrow[dl, phantom, "\subseteq"] \\
\mathbb{N}_0 & \mathbb{N}_0 \arrow[l, "\Phi_{even}"]
\end{tikzcd}$$

Proof obligations in the PF-style

In general:

Input/output property preservation (functions)

Proof obligation

$$\langle \forall\ x\ :\ p\ x :\ q\ (f\ x) \rangle \tag{69}$$

stating that function f **ensures** *property* q *on its* **output** *every time property* p *holds on its* **input** *PF-transforms to*

$$f \cdot \Phi_p \subseteq \Phi_q \cdot f \quad \textit{cf. diagram} \quad \begin{array}{ccc} A & \xleftarrow{\Phi_p} & A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xleftarrow[\Phi_q]{} & B \end{array} \tag{70}$$

Predicates as "types"

We will write "type declaration"

$$\Phi_q \xleftarrow{f} \Phi_p \tag{71}$$

to mean (70).

Exercise 16: Show that (70) and

$$f \cdot \Phi_p \subseteq \Phi_q \cdot \top \tag{72}$$

are the same.

□

Exercise 17: Prove the equivalence

$$\Phi_q \xleftarrow{id} \Phi_p \quad \Leftrightarrow \quad q \Leftarrow p \tag{73}$$

□

Exercises

Exercise 18: Infer from (71) and properties (37) to (48) the following ESC (*extended static checking*) properties:

$$\Phi_q \xleftarrow{f} \Phi_{p_1} \cup \Phi_{p_2} \quad \Leftrightarrow \quad \Phi_q \xleftarrow{f} \Phi_{p_1} \ \wedge \ \Phi_q \xleftarrow{f} \Phi_{p_2} \tag{74}$$

$$\Phi_{q_1} \cdot \Phi_{q_2} \xleftarrow{f} \Phi_p \quad \Leftrightarrow \quad \Phi_{q_1} \xleftarrow{f} \Phi_p \ \wedge \ \Phi_{q_2} \xleftarrow{f} \Phi_p \tag{75}$$

□

Exercise 19: Using (72) and the relational version of McCarthy's conditional combinator which follows,

$$c \rightarrow f, g \ = \ f \cdot \Phi_c \cup g \cdot \Phi_{\neg c} \tag{76}$$

infer the *conditional ESC* rule which follows:

$$\Phi_q \xleftarrow{c \rightarrow f, g} \Phi_p \quad \Leftrightarrow \quad \Phi_q \xleftarrow{f} \Phi_p \cdot \Phi_c \ \wedge \ \Phi_q \xleftarrow{g} \Phi_p \cdot \Phi_{\neg c} \tag{77}$$

□

Relationship with Hoare Logic

Let us show that **Hoare triples** such as

$$\{p\}P\{q\} \tag{78}$$

are also instances of ESC proof obligations. First we spell out the meaning of (78):

$$\langle \forall\ s\ :\ p\ s :\ \langle \forall\ s'\ :\ s \xrightarrow{P} s'\ :\ q\ s' \rangle\rangle \tag{79}$$

Then (recording the meaning of program P as relation $[\![P]\!]$ on program states) we PF-transform (79) into

$$\Phi_p \subseteq [\![P]\!] \setminus (\Phi_q \cdot \top) \tag{80}$$

thanks to the introduction of relational (left) **division**,

$$b\ (R \setminus S)\ a\ \Leftrightarrow\ \langle \forall\ c\ :\ c\ R\ b :\ c\ S\ a \rangle \tag{81}$$

Relationship with Hoare Logic

Thanks to “al-djabr” rule

$$R \cdot X \subseteq S \quad \Leftrightarrow \quad X \subseteq R \setminus S \tag{82}$$

we obtain

$$[\![P]\!] \cdot \Phi_p \subseteq \Phi_q \cdot \top \tag{83}$$

equivalent to

$$[\![P]\!] \cdot \Phi_p \subseteq \Phi_q \cdot [\![P]\!]$$

which shares the same scheme as

$$f \cdot \Phi_p \ \subseteq \ \Phi_q \cdot f$$

earlier on.

Summary

In general, we will write "type declaration"

$$\Psi \xleftarrow{R} \Phi \tag{84}$$

to mean

$$R \cdot \Phi \subseteq \Psi \cdot R \tag{85}$$

In words:

- Notation (84) can be regarded as the **type assertion** that, if fed with values (or starting on states) "of type Φ" computation R yields results (moves to states) "of type Ψ" (if it terminates).
- So functional ESC POs and Hoare triples are one and the same device: a way to **type** computations, be them specified as (always terminating, deterministic) functions or encoded into (possibly non-terminating, non-deterministic) programs.

Background — two useful coreflexives

Domain:

$$\delta\, R \quad \triangleq \quad \ker R \cap id \tag{86}$$

"Al-djabr" rule:

$$\delta\, R \subseteq \Phi \quad \Leftrightarrow \quad R \subseteq \top \cdot \Phi \tag{87}$$

Range:

$$\rho\, R \quad \triangleq \quad \mathrm{img}\, R \cap id \tag{88}$$

"Al-djabr" rule:

$$\rho\, R \subseteq \Phi \quad \Leftrightarrow \quad R \subseteq \Phi \cdot \top \tag{89}$$

Relating coreflexives with conditions

Domain/range elimination:

$$\top \cdot \delta R = \top \cdot R \tag{90}$$

$$\rho R \cdot \top = R \cdot \top \tag{91}$$

Mapping back and forward:

$$\Phi \subseteq \Psi \Leftrightarrow \Phi \subseteq \top \cdot \Psi \tag{92}$$

Closure properties:

$$R \cdot \Phi \subseteq S \Leftrightarrow R \cdot \Phi \subseteq S \cdot \Phi \tag{93}$$

$$\Phi \cdot R \subseteq S \Leftrightarrow \Phi \cdot R \subseteq \Phi \cdot S \tag{94}$$

Exercise 20: Show that

$$\delta R \subseteq \delta S \Leftrightarrow R \subseteq \top \cdot S \tag{95}$$

holds.

□

PO discharge by calculation

We close the lecture by discharging proof obligation (54) once captured by type diagram

$$\Phi_{even} \xleftarrow{twice} \Phi_{even}$$

We reason:

$$
\begin{array}{cl}
 & \Phi_{even} \xleftarrow{twice} \Phi_{even} \\
\Leftrightarrow & \{ \ (71) \ \} \\
 & twice \cdot \Phi_{even} \subseteq \Phi_{even} \cdot twice \\
\Leftrightarrow & \{ \ \text{definition of } even \ \} \\
 & twice \cdot \Phi_{even} \subseteq \rho\, twice \cdot twice \\
\Leftrightarrow & \{ \ R = (\rho\, R) \cdot R \ \} \\
 & twice \cdot \Phi_{even} \subseteq twice \\
\Leftrightarrow & \{ \ \text{"al-djabr"} \ \} \\
 & \Phi_{even} \subseteq twice^{\circ} \cdot twice \\
\Leftrightarrow & \{ \ \text{func. kernels are reflexive} \ \} \\
 & \mathrm{TRUE}
\end{array}
$$

Fourth lecture

Schedule: Tuesday Feb 26th, 17h20-18h10

Learning outcomes:

- Discharging proof obligations via PF-transform. Pre/post conditions. Invariants.
- Extended static checking in the PF-style. PF-calculation of weakest pre-conditions for invariant preservation.

Calculating Invariants and Preconditions

- Wherever a function f does not ensure preservation of invariant inv, there is always a **pre-condition** pre which enforces this at the cost of *partializing* f.
- In the limit, pre is the everywhere false predicate.
- As a rule, the average programmer will become aware of such a pre-condition at runtime, in the **testing** phase.
- One can find it much earlier, at specification time, when trying to discharge the standard proof obligation

$$\langle \forall\ a\ :\ inv\ a :\ inv (f\ a) \rangle \tag{96}$$

which then extends to

$$\langle \forall\ a\ :\ pre\ a \wedge inv\ a :\ inv (f\ a) \rangle \tag{97}$$

PF-ESC instead of invent & verify

However,

- Bound to **invent** *pre*, we'll hope to have guessed the **weakest** such pre-condition. Otherwise, future use of f will be spuriously constrained.
- Can we be sure of having hit the **weakest** pre-condition?

Our approach (**PF-ESC**) will be as follows:

- We take the PF-transform of $inv(f\ a)$ in (97) — at data level — and attempt to rewrite it to a term involving $inv\ a$ and possibly "something else": the **calculated** pre-condition.
- This will be the *weakest* provided the calculation stays within equivalence steps (as shown in the next slides).

Weakest pre-conditions

Let us transform (71) according to the PF-calculus studied so far:

$$
\begin{array}{cl}
& \Phi_q \xleftarrow{f} \Phi_p \\
\Leftrightarrow & \{ \ (72) \ \} \\
& f \cdot \Phi_p \subseteq \Phi_q \cdot \top \\
\Leftrightarrow & \{ \ (89) \ \} \\
& \rho\,(f \cdot \Phi_p) \subseteq \Phi_q
\end{array}
$$

On the other hand,

$$
\begin{array}{cl}
& f \cdot \Phi_p \subseteq \Phi_q \cdot \top \\
\Leftrightarrow & \{ \ (37) \ \} \\
& \Phi_p \subseteq f^\circ \cdot \Phi_q \cdot \top \\
\Leftrightarrow & \{ \ \text{coreflexives} \ \} \\
& \Phi_p \subseteq \top \cdot \Phi_q \cdot f
\end{array}
$$

Weakest pre-conditions

Putting everything together, from $\Phi_q \xleftarrow{f} \Phi_p$ we obtain GC

$$\underbrace{\rho\,(f \cdot \Phi_p)}_{\substack{\text{strongest}\\ \text{post-condition}}} \subseteq \Phi_q \quad \Leftrightarrow \quad \Phi_p \subseteq \underbrace{\top \cdot \Phi_q \cdot f}_{\substack{\text{weakest}\\ \text{pre-condition}}} \tag{98}$$

Back to (97), to obtain the weakest pre-condition *pre* for f to preserve invariant *inv*, we just have to factorize the overall WP over *inv* on the input:

$$\underbrace{\top \cdot \Phi_{inv} \cdot f}_{WP} \quad = \quad \Phi_{pre} \cdot \Phi_{inv} \tag{99}$$

Back to points, this means converting (97) into an equivalence

$$\langle \forall\ a\ ::\ (pre\ \ a) \wedge (inv\ \ a) \Leftrightarrow inv(f\ \ a) \rangle \tag{100}$$

In summary: $\Phi_{pre} \cdot \Phi_{inv}$ is not only sufficient but also necessary.

Background

In general, the **weakest** (liberal) **pre-condition operator** is the upper adjoint of the following "al-djabr" rule which combines two already seen — range (89) and left division (81):

$$\rho\,(R \cdot \Phi) \subseteq \Psi \quad \Leftrightarrow \quad \Psi \xleftarrow{R} \Phi \quad \Leftrightarrow \quad \Phi \subseteq \underbrace{R \setminus (\Psi \cdot \top)}_{R \mathbin{\rightthreetimes} \Psi} \tag{101}$$

Notation $R \mathbin{\rightthreetimes} \Psi$ is taken from [2]. The pointwise version $wlp\ R\ \psi$ of $R \mathbin{\rightthreetimes} \Psi$ is:

$$wlp\ R\ \psi \quad \triangleq \quad \langle \bigvee\ \phi\ :\ \langle \forall\ b, a\ :\ b\ R\ a \wedge \phi\ a :\ \psi\ b \rangle\ :\ p \rangle$$

Case study 1: PF-ESC at work

We want to calculate the WP for

$$add\ x\ l \quad \triangleq \quad a : l$$

to preserve the **no duplicates** invariant on finite lists.

- First step: PF-transform $X^\star$ to $\mathbb{N} \rightharpoonup X$ (**simple** relation telling which elements take which position in list).

  Then the **no duplicates** invariant on L is encoded as $\ker L \subseteq id$ (L is injective)

  Finally, $add\ x\ L$ PF-transforms to

$$\underline{x} \cdot \underline{1}^\circ \cup L \cdot succ^\circ \tag{102}$$

  cf. back to points: $\{1 \mapsto x\} \cup \{i + 1 \mapsto (L\ i) : i \leftarrow \delta\ L\}$.

Case study 1: PF-ESC at work

- Second step: we start from the right hand side $inv(add\ x\ L)$ of (100) and re-write it by successive equivalence steps until we reach:
  - condition $inv\ l$ ...
  - ... "plus something else" — the calculated weakest pre-condition.
- Since the PF-transformed proof has to do with injectivity of union of relations, the following fact

$$R \cup S \text{ is injective } \Leftrightarrow$$
$$R \text{ is injective} \wedge S \text{ is injective} \wedge R^{\circ} \cdot S \subseteq id \quad (103)$$

(easy to prove) is likely to be of use.

Case study 1: PF-ESC at work

$add\ x\ L$ has no duplicates

$\Leftrightarrow$ { cf. (102) etc }

$\underline{x} \cdot \underline{1}^\circ \cup L \cdot succ^\circ$ is injective

$\Leftrightarrow$ { (103) }

$\underline{x} \cdot \underline{1}^\circ$ is injective $\wedge\ L \cdot succ^\circ$ is injective $\wedge\ (\underline{x} \cdot \underline{1}^\circ)^\circ \cdot L \cdot succ^\circ \subseteq id$

$\Leftrightarrow$ { definition of injective (twice) ; "al-djabr" (37) }

$\underline{1} \cdot \underline{x}^\circ \cdot \underline{x} \cdot \underline{1}^\circ \subseteq id \wedge succ \cdot L^\circ \cdot L \cdot succ^\circ \subseteq id \wedge \underline{x}^\circ \cdot L \subseteq \underline{1}^\circ \cdot succ$

$\Leftrightarrow$ { "al-djabr" (37,38) as much as possible }

$\underline{x}^\circ \cdot \underline{x} \subseteq \underline{1}^\circ \cdot \underline{1} \wedge L^\circ \cdot L \subseteq succ^\circ \cdot succ \wedge \underline{x}^\circ \cdot L \subseteq \underline{1}^\circ \cdot succ$

$\Leftrightarrow$ { kernel of constant function is $\top$; *succ* is an injection }

$\textsc{True} \wedge L^\circ \cdot L \subseteq id \wedge \underline{x}^\circ \cdot L \subseteq \underline{1}^\circ \cdot succ$

Case study 1: summary

We have thus calculated:

$$\mathit{add}\ x\ L \text{ has no duplicates} \quad \Leftrightarrow \quad \underbrace{L \text{ is injective}}_{\text{no duplicates in } L} \ \wedge\ \underbrace{\underline{x}^{\circ} \cdot L \subseteq \underline{1}^{\circ} \cdot \mathit{succ}}_{\text{WP}}$$

PW-expansion of the calculated WP:

$$\begin{array}{ll}
 & \underline{x}^{\circ} \cdot L \subseteq \underline{1}^{\circ} \cdot \mathit{succ} \\
\Leftrightarrow & \qquad \{ \ \text{go pointwise: (30) twice} \ \} \\
 & \langle \forall\ n\ ::\ x\ L\ n \Rightarrow 1 = 1 + n \rangle \\
\Leftrightarrow & \qquad \{ \ L \text{ models list } l \ \} \\
 & \langle \forall\ n\ :\ n \in \mathsf{inds}\, l\ :\ x = (l\ n) \Rightarrow 1 = 1 + n \rangle \\
\Leftrightarrow & \qquad \{ \ 1 = 1 + n \text{ always false } (n \in \mathbb{N}) \ \} \\
 & \langle \forall\ n\ :\ n \in \mathsf{inds}\, l\ :\ (l\ n) \neq x \rangle
\end{array}$$

Case study 2: PF-ESC at work

From the mobile phone directory problem we select preservation of the no duplicates invariant by function

$$store\ c \quad \triangleq \quad (take\ 10) \cdot (c :) \cdot filter(c \neq)$$

Remarks:

- It's sufficient to show that $(c :) \cdot filter(c \neq)$ preserves injectivity, since $take\ n\ L \ \subseteq \ L$ $(\forall n)$ and *smaller than injective is injective*
- Defined over PF-transformed lists, *filter* becomes

$$filter(c \neq)L \quad \triangleq \quad (\neg\rho\ \underline{c}) \cdot L \tag{104}$$

where the negated range operator $(\neg\rho)$ satisfies property

$$\Phi \subseteq \neg\rho\, R \quad \Leftrightarrow \quad \Phi \cdot R \subseteq \bot \tag{105}$$

Case study 2: PF-ESC at work

$$
\begin{array}{ll}
 & c : (\mathit{filter}(c \neq) L) \text{ is injective} \\
\Leftrightarrow & \qquad \{ \text{ case study 1, (104) } \} \\
 & (\neg \rho\, \underline{c}) \cdot L \text{ is injective} \wedge \underline{c}^\circ \cdot (\neg \rho\, \underline{c}) \cdot L \subseteq \underline{1}^\circ \cdot \mathit{succ} \\
\Leftarrow & \qquad \{ \text{ smaller than injective is injective } \} \\
 & L \text{ is injective} \wedge \underline{c}^\circ \cdot (\neg \rho\, \underline{c}) \cdot L \subseteq \underline{1}^\circ \cdot \mathit{succ} \\
\Leftrightarrow & \qquad \{ \text{ converses } \} \\
 & L \text{ is injective} \wedge L^\circ \cdot (\neg \rho\, \underline{c}) \cdot \underline{c} \subseteq \mathit{succ}^\circ \cdot \underline{1} \\
\Leftrightarrow & \qquad \{ (\neg \rho\, \underline{c}) \cdot \underline{c} = \bot \text{ by left-cancellation of (105) } \} \\
 & L \text{ is injective} \wedge L^\circ \cdot \bot \subseteq \mathit{succ}^\circ \cdot \underline{1} \\
\Leftrightarrow & \qquad \{ \text{ bottom is below anything } \} \\
 & L \text{ is injective} \wedge \textsc{True}
\end{array}
$$

Case study 2: PF-ESC at work

Moral of this case study:

> *Although the implication in the second step of the reasoning could put weakness of calculated pre-condition at risk, we've calculated the weakest of all conditions anyway* (TRUE)*.*

Exercise 21: Show that (105) stems from "al-djabr" rule

$$\Phi \subseteq \neg\delta\, R \quad \Leftrightarrow \quad R \subseteq \bot/\Phi \tag{106}$$

among others.

□

Fifth lecture

Schedule: Wednesday Feb 27th, 15h00-15h50

Learning outcomes:

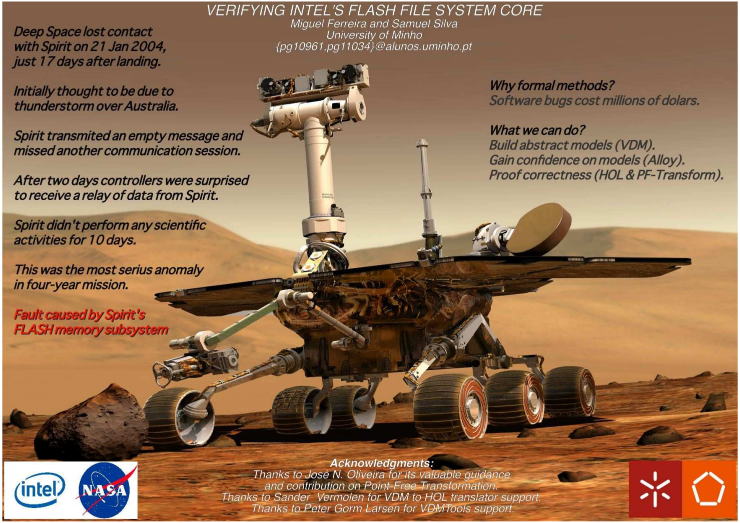
- Proof obligations in-the-large and in-the-small. Thinking big writing less.
- The VFS (Verified File System) case study.
- The broad picture: integration with theorem provers and model checkers
- Thinking "bigger" : invariants as coreflexive bisimulations in a coalgebraic setting.

Case study 3: Verified File System

A real-life case study:

- **VSR** (Verified Software Repository) initiative
- **VFS** (Verified File System) on Flash Memory — challenge put forward by Rajeev Joshi and Gerard Holzmann (NASA JPL) [7]
- Two levels — POSIX level and (NAND) flash level
- Working document: **Intel ® Flash File System Core Reference Guide** (Oct. 2004) is POSIX aware.

Case study 3: Verified File System



Case study 3: Verified File System

The problem (sample):

File System API Reference

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

Verified File System Project

Sample of model's data types (simplified):

$$System = \{table : OpenFileDescriptorTable, tar : Tar\}$$
$$\textbf{inv} \;\; sys \triangleq \langle \forall\, ofd \;:\; ofd \in rng\ (table\ sys) :\; path\ ofd \in dom\ tar\ sys \rangle$$

where

$$OpenFileDescriptorTable = FileHandler \rightharpoonup OpenFileDescriptor$$

$$Tar = Path \rightharpoonup File$$
$$\textbf{inv} \;\; tar \triangleq \langle \forall\, p \;:\; p \in dom\ tar :\; dirName(p) \in dom\ tar \land fileType(attributes(tar(dirName\ p))) = Directory \rangle$$

$$OpenFileDescriptor = \{path : Path, \ldots\}$$

Verified File System Project

(Sample) API function:

$$FS\_DeleteFileDir : Path \rightarrow System \rightarrow (System \times FFS\_Status)$$

$$FS\_DeleteFileDir\ p\ sys \triangleq$$

$$\begin{aligned}&\mathit{if}\ \ p \neq Root \wedge p \in dom\ (tar\ \ sys) \wedge \text{pre-}FS\_DeleteFileDir\_System\ p\ sys\\ &\mathit{then}\ (FS\_DeleteFileDir\_System\ p\ sys, FFS\_StatusSuccess)\\ &\mathit{else}\ (sys, FS\_DeleteFileDir\_Exception\ p\ sys)\end{aligned}$$

where

$$FS\_DeleteFileDir\_System : Path \rightarrow System \rightarrow System$$

$$FS\_DeleteFileDir\_System\ p\ (h, t) \triangleq$$

$$(h, FS\_DeleteFileDir\_Tar\ \{p\}\ t)$$

$$\textbf{pre} \left\langle \begin{array}{c} \forall\ buffer \\ buffer \in rng\ \ h : \\ path\ \ buffer \neq p \wedge \text{pre-}FS\_DeleteFileDir\_Tar\ \{p\}\ t \end{array} \right\rangle$$

Verified File System Project

Sample API function (continued):

$$FS\_DeleteFileDir\_Tar : \mathcal{P}\,Path \to Tar \to Tar$$
$$FS\_DeleteFileDir\_Tar\ s\ t \triangleq\ tar \setminus s$$
$$\textbf{pre}\ \langle \forall\ p\ :\ p \in dom\ tar :\ dirName\ p \in s \Rightarrow p \in s \rangle;$$

where

$$dirName : Path \to Path$$
$$\begin{aligned} dirName\ p \triangleq\ & if\ \ p = Root \vee len\ \ p = 1 \\ & \quad then\ \ Root \\ & \quad else\ \ blast\ \ p \end{aligned}$$

and so on. (**NB:** *blast* selects all but the last element of a list.)

Invariant structural synthesis (coreflexives)

- Real-size problems show **where complexity is**, namely the intricate structure involving nested datatype invariants.
- Need to calculate the associated coreflexives.
- Denoting by F_p the fact that data type constructor F is constrained by invariant p, we will write $\in_{\mathsf{F}_p}$ to denote the coreflexive which captures *all* constraints involved in declaring F_p, calculated by induction on the structure of types:

$$\in_{\mathsf{F}_p} = (\in_{\mathsf{F}}) \cdot \Phi_p \quad (107)$$

$$\in_{\mathsf{K}} = id \quad (108)$$

$$\in_{\mathsf{Id}} = id \quad (109)$$

$$\in_{\mathsf{F}\times\mathsf{G}} = \in_{\mathsf{F}} \times \in_{\mathsf{G}} \quad (110)$$

$$\in_{\mathsf{F}+\mathsf{G}} = \in_{\mathsf{F}} + \in_{\mathsf{G}} \quad (111)$$

$$\in_{\mathsf{F}\cdot\mathsf{G}} = \mathsf{F}(\in_{\mathsf{G}}) \quad (112)$$

Invariant structural synthesis (coreflexives)

Example:

$$
\begin{array}{cl}
& \in_{System} \\
= & \qquad \{ \text{ definition of } System \text{ } \} \\
& \in_{(OpenFileDescriptorTable \times FStore)_{ri}} \\
= & \qquad \{ \text{ (107) and datatype definitions } \} \\
& (\in_{FileHandler \rightharpoonup OpenFileDescriptor} \times \in_{(Path \rightharpoonup File)_{pc}}) \cdot \Phi_{ri} \\
= & \qquad \{ \text{ (108) and (107) } \} \\
& (id \times \in_{Path \rightharpoonup File} \cdot \Phi_{pc}) \cdot \Phi_{ri} \\
= & \qquad \{ \text{ (108) } \} \\
& (id \times \Phi_{pc}) \cdot \Phi_{ri}
\end{array}
\tag{113}
$$

where we abbreviate *System*'s invariant by predicate *ri* (for "referential integrity") and *FStore*'s invariant by *pc* (for "paths closed"):

Facing complexity

Need to "find structure" in the specification text:

- *FS_DeleteFileDir p* has conditional "shape"

$$c \longrightarrow \langle f \cdot \Phi_p, \underline{k} \rangle, \langle id, g \rangle \tag{114}$$

where

- c is the (main) if-then-else's condition
- f abbreviates *FS_DeleteFileDir_System p*
- p is the precondition of f
- k abbreviates *FFS_StatusSuccess*
- g abbreviates *FS_DeleteFileDir_Exception p*

What's the advantage of pattern (114)?

See the "divide and conquer" rules which follow:

Breaking complexity of POs

Further to (73), (75), (77):

- **Trivial**:

$$id \xleftarrow{R} \Phi \quad \Leftrightarrow \quad \textsc{True} \quad \Leftrightarrow \quad \Phi \xleftarrow{R} \perp \tag{115}$$

- **Trading**:

$$\Upsilon \xleftarrow{R} \Phi \cdot \Psi \quad \Leftrightarrow \quad \Upsilon \xleftarrow{R \cdot \Phi} \Psi \tag{116}$$

- **Composition** (**Fusion**):

$$\Psi \xleftarrow{R \cdot S} \Phi \quad \Leftarrow \quad \Psi \xleftarrow{R} \Upsilon \wedge \Upsilon \xleftarrow{S} \Phi \tag{117}$$

Breaking complexity of POs

- **Split by conjunction**:

$$\Psi_1 \cdot \Psi_2 \xleftarrow{R} \Phi \quad \Leftrightarrow \quad \Psi_1 \xleftarrow{R} \Phi \;\wedge\; \Psi_2 \xleftarrow{R} \Phi \tag{118}$$

— generalizes (75)

- **Weakening/strengthening**:

$$\Psi \xleftarrow{R} \Phi \quad \Leftarrow \quad \Psi \supseteq \Theta \wedge \;\Theta \xleftarrow{R} \Upsilon \;\wedge \Upsilon \supseteq \Phi \tag{119}$$

- **Separation**:

$$\Upsilon \cdot \Theta \xleftarrow{R} \Phi \cdot \Psi \quad \Leftarrow \quad \Upsilon \xleftarrow{R} \Phi \;\wedge\; \Theta \xleftarrow{R} \Psi \tag{120}$$

— outcome of (119), (118)

Breaking complexity of POs

- **Splitting** (functions):

$$\Psi \times \Upsilon \xleftarrow{\langle f,g \rangle} \Phi \quad \Leftrightarrow \quad \Psi \xleftarrow{f} \Phi \ \wedge \ \Upsilon \xleftarrow{g} \Phi \qquad (121)$$

- **Splitting** (in general):

$$\Psi \times \Upsilon \xleftarrow{\langle R,S \rangle} \Phi \quad \Leftrightarrow \quad \Psi \xleftarrow{R} \Phi \cdot \delta\, S \ \wedge \ \Upsilon \xleftarrow{S} \Phi \cdot \delta\, R \qquad (122)$$

- **Product**:

$$\Phi' \times \Psi' \xleftarrow{R \times S} \Phi \times \Psi \quad \Leftrightarrow \quad \Phi' \xleftarrow{R} \Phi \ \wedge \ \Psi' \xleftarrow{S} \Psi \qquad (123)$$

Breaking complexity of POs

- **Conditional**:

$$\Psi \xleftarrow{c \to R,S} \Phi \quad \Leftrightarrow \quad \Psi \xleftarrow{R} \Phi \cdot \Phi_c \;\wedge\; \Psi \xleftarrow{S} \Phi \cdot \Phi_{\neg c} \quad (124)$$

which generalizes (74).

NB:

- Close relationship with Hoare logic axioms

— but note many **equivalences** instead of implications

Exercise 22: Use the PF-calculus to prove the correctness of the rules given above.

□

Verified File System Project

Checking *FS_DeleteFileDir*:

$$\in_{System \times FFS\_Status} \xleftarrow{FS\_DeleteFileDir\ \ p} \in_{System}$$

$$\Leftrightarrow \quad \{ \ (114) \ \}$$

$$\in_{System} \times id \xleftarrow{c \rightarrow \langle f \cdot \Phi_p, \underline{k} \rangle, \langle id, g \rangle} \in_{System}$$

$$\Leftrightarrow \quad \{ \ \text{conditional (124)} \ \}$$

$$\in_{System} \times id \xleftarrow{\langle f \cdot \Phi_p, \underline{k} \rangle} \in_{System} \cdot \Phi_c$$

$$\wedge$$

$$\in_{System} \times id \xleftarrow{\langle id, g \rangle} \in_{System} \cdot \Phi_{\neg c}$$

Verified File System Project

$\Leftrightarrow$ { splitting (122,121) }

$$\text{€}_{System} \xleftarrow{f \cdot \Phi_p} \text{€}_{System} \cdot \Phi_c$$

$$\wedge$$

$$id \xleftarrow{\underline{k}} \text{€}_{System} \cdot \Phi_c \cdot \delta\,(f \cdot \Phi_p)$$

$$\wedge$$

$$\text{€}_{System} \xleftarrow{id} \text{€}_{System} \cdot \Phi_{\neg c}$$

$$\wedge$$

$$id \xleftarrow{g} \text{€}_{System} \cdot \Phi_{\neg c}$$

$\Leftrightarrow$ { (115), (73) }

Case study 3: Verified File System

$$\in_{System} \xleftarrow{f \cdot \Phi_p} \in_{System} \cdot \Phi_c$$

$\Leftrightarrow$ { trading (116), unfold $\in_{System}$ (113) }

$$(id \times \Phi_{pc}) \cdot \Phi_{ri} \xleftarrow{f \cdot \Phi_p \cdot \Phi_c} (id \times \Phi_{pc}) \cdot \Phi_{ri}$$

$\Leftarrow$ { separating (120) }

$$\Phi_{ri} \xleftarrow{f \cdot \Phi_p \cdot \Phi_c} \Phi_{ri} \quad \wedge \quad id \times \Phi_{pc} \xleftarrow{f \cdot \Phi_p \cdot \Phi_c} id \times \Phi_{pc}$$

$\Leftrightarrow$ { trading (116) and implication $c \Rightarrow p$ }

$$\Phi_{ri} \xleftarrow{f} \Phi_{ri} \cdot \Phi_c \qquad \wedge$$
$$id \times \Phi_{pc} \xleftarrow{f} (id \times \Phi_{pc}) \cdot \Phi_c$$

Case study 3: Verified File System

- So much for PO calculation "in-the-large".
- Going "in-the-small" means spelling out invariants, functions and pre-conditions and reason as in the previous case studies
- Let us pick the first PO, $\Phi_{ri} \xleftarrow{f} \Phi_{ri} \cdot \Phi_c$, for example.
- As earlier on, we go pointwise and try to rewrite $ri(f(M,N))$ — M keeps open file descriptors, N the file contents — into $ri(M,N)$ + a weakest precondition; then we compare the outcome with what the designer wrote (Φ_c).

Case study 3: Verified File System

Clearly,

$$ri(M,N) \quad \triangleq \quad \rho\,(path \cdot M) \subseteq \delta\, N$$

cf. diagram

$$\begin{tikzcd} & FileHandler \arrow[r, "M"] & OpenFileDescriptor \arrow[dl, "path"] \\ Path \arrow[rr, "N"'] & & File \end{tikzcd}$$

is a referential integrity constraint relating paths in open-file descriptors and paths in the file store N. PF calculation will lead to

$$ri(M,N) \ \triangleq \ \ path \cdot M \subseteq N^{\circ} \cdot \top \tag{125}$$

Case study 3: Verified File System

On the other hand, $f(M, N)$ — that is $FS\_DeleteFileDir\_System\ p\ (M, N)$ — PF-transforms to $(M, N \cdot \neg\rho\ \underline{p})$. Generalizing from single paths to sets S of paths:

$$
\begin{array}{ll}
& ri(M, N \cdot \Phi_{\neg S}) \\
\Leftrightarrow & \qquad \{\ (125)\ \} \\
& path \cdot M \subseteq (N \cdot \Phi_{\neg S})^{\circ} \cdot \top \\
\Leftrightarrow & \qquad \{\ \text{converses (27,28, 65)}\ \} \\
& path \cdot M \subseteq \Phi_{\neg S} \cdot N^{\circ} \cdot \top \\
\Leftrightarrow & \qquad \{\ \text{(91), coreflexives (66), } (\cdot\top) \text{ distribution}\ \} \\
& path \cdot M \subseteq \Phi_{\neg S} \cdot \top \cap N^{\circ} \cdot \top \\
\Leftrightarrow & \qquad \{\ \cap\text{-universal (42)}\ \}
\end{array}
$$

Case study 3: Verified File System

$$path \cdot M \subseteq \Phi_{\neg S} \cdot \top \wedge path \cdot M \subseteq N^{\circ} \cdot \top$$

$$\Leftrightarrow \qquad \{ \text{ "al-djabr" ; (125) } \}$$

$$\underbrace{M \subseteq path^{\circ} \cdot \Phi_{\neg S} \cdot \top}_{\text{wp}} \;\wedge\; ri(M, N)$$

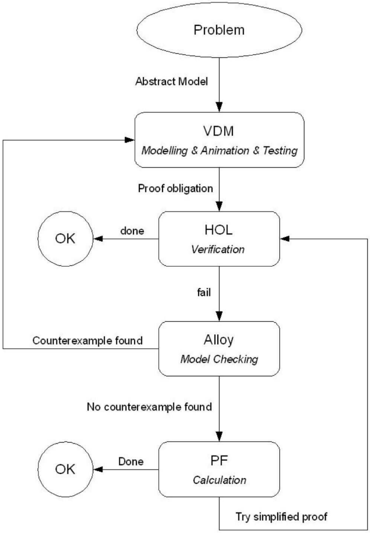
$$\Leftrightarrow \qquad \{ \text{ going pointwise } \}$$

$$\langle \forall\, b \;:\; b \in rng\ M :\; path\ b \notin S \rangle \wedge ri(M, N)$$

Summary:

- Thus we've checked (part) of the pre-condition. The other checks are performed in a similar way. (See **Addendum**, slide 156.)
- Two levels of PO calculation: **in-the-large** (PO level) and **in-the-small** (where PF-notation describes data).
- PO-level useful in preparing POs for a theorem prover, see diagram which follows:

Broad picture: a “all-in-one” strategy



Final comments

- Algebra of POs bridges Hoare logic and type theory
- Close (formal) relationship with similar work in PF **data dependency** theory [10], cf. $\Phi \xrightarrow{R} \Psi$ with

$$f \overset{R}{\rightarrow} g$$

where R models a set (eg. of tuples) and f and g are observations (eg. sets of attributes), meaning

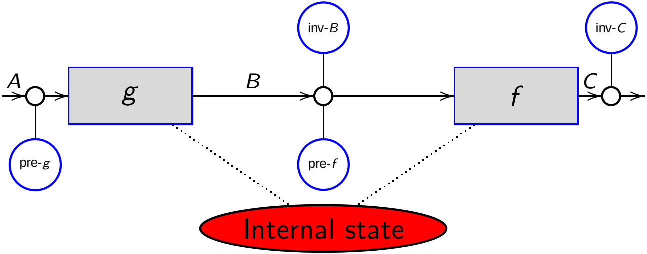
$$\langle \forall\ t, t' \ :\ t, t' \in R :\quad f\ t = f\ t' \quad \Rightarrow \quad g\ t = g\ t'\ \rangle$$

Compare, for instance, (119) with the **Decomposition** axiom of FDs:

$$h \overset{R}{\rightarrow} k \quad \Leftarrow \quad h \geq f \quad \wedge \quad f \overset{R}{\rightarrow} g \quad \wedge \quad g \geq k$$

The "broad picture"

- Software systems are far more complex than API functions.
- In component-oriented design the *programming unit* is the **component**, not the function.
- Components have **state**:

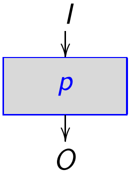


- The concept of **invariant**, for instance, makes sense relative to the whole component, not just a particular function.

The "broad picture"

A (generic) component p with input interface I and output interface O

$$p : O \longleftarrow I$$



is a Mealy machine

$$\mathsf{B}(U_p \times O) \xleftarrow{\;p\;} U_p \times I \qquad (126)$$

where U_p is the internal state and monad B captures a particular behaviour pattern (eg. powerset for non-deterministic behaviour).

Current work

- The concept of a **coalgebra** is a very convenient formal device for characterizing software components.
- "Currying" mediates Mealy machines and (a special class of) coalgebras, cf.

$$\underbrace{\mathsf{B}(U_p \times O)^I}_{\mathsf{F}\,U_p} \xleftarrow{\;c\;} U_p \qquad (127)$$

- Elsewhere [9] we have shown that **invariants** are special cases of **bisimulations**, which are written $\mathsf{F}R \xleftarrow{\;c\;} R$ for coalgebra c of "shape" F.
- Currently working (with **Luís Barbosa** and **Alexandra Silva**) on scaling-up PO-reasoning from function to component level.
- Joint work with **Claudia Necco** and **Joost Visser** on automating PF-calculation [8].

Current work

"Predicates as types" view carries over universal constructs, for instance functional **products** — recall (121):

$$
\begin{tikzcd}
\Psi & \Psi \times \Upsilon \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & \Upsilon \\
 & \Phi \arrow[ul, "f"] \arrow[u, "\langle f,g \rangle"] \arrow[ur, "g"'] &
\end{tikzcd}
\tag{128}
$$

Note that the POs associated to the projections,

$$
\pi_1 \cdot (\Psi \times \Upsilon) \subseteq \Psi \cdot \pi_1
$$
$$
\pi_2 \cdot (\Psi \times \Upsilon) \subseteq \Upsilon \cdot \pi_2
$$

are (PF-transformed) instances of the **theorems for free** [15] of the corresponding (**polymorphic**) types. (Nothing to prove )

Current work

"Predicates as types" view carries over folds/unfolds etc. For instance, let us check the diagram of a **fold**:

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[d, "(\!|R|\!)"'] & \mathsf{F}(\mu\mathsf{F}) \arrow[l, "in"'] \arrow[d, "\mathsf{F}(\!|R|\!)"] \\
\Phi_p & \mathsf{F}\,\Phi_p \arrow[l, "R"]
\end{tikzcd}
$$

$$
\begin{array}{ll}
& \Phi_p \xleftarrow{(\!|R|\!)} \mu\mathsf{F} \\
\Leftrightarrow & \quad \{ \text{ definition (84) } \} \\
& (\!|R|\!) \subseteq \Phi_p \cdot (\!|R|\!) \\
\Leftarrow & \quad \{ \text{ relational cata-fusion (129) } \} \\
& R \cdot \mathsf{F}\,\Phi_p \subseteq \Phi_p \cdot R \\
\Leftrightarrow & \quad \{ \text{ definition (84) } \} \\
& \Phi_p \xleftarrow{R} \mathsf{F}\,\Phi_p
\end{array}
$$

$$
(\!|T|\!) \subseteq S \cdot (\!|R|\!) \quad \Leftarrow \quad T \cdot \mathsf{F}\,S \subseteq S \cdot R \qquad (129)
$$

About the background

- This tutorial finds its roots in the **excellent** background for CS research developed by the **MPC** (Mathematics of Program Construction) group [1, 6, 3]
- For a good textbook on relation algebra, examples and applications see [5]
- For other experiments on the PF-transform applied to different CS theories see eg.
  - Data dependency theory (databases) [10]
  - Hashing [11]
  - Algebraic/coalgebraic refinement [12, 4]
  - Bisimulations [9]
  - Separation logic [14]

Closing

PF-transform "road-map":

- The PF-transform is applicable to CS theories whose base concepts are defined by complex pointwise formulæ — the "**hard** problem"
- The *tradition* is to develop a (PW) axiomatic theory able to do without the complex semantic model
- The PF-transform not only makes the validation of such theory much simpler but also makes **direct** reasoning in the model viable
- Last point particularly useful in the case of incomplete theories (eg. separation logic)

Don't hesitate: join the PF-community and start "thinking big" 

Closing

PF-transform "road-map":

- The PF-transform is applicable to CS theories whose base concepts are defined by complex pointwise formulæ — the "**hard** problem"
- The *tradition* is to develop a (PW) axiomatic theory able to do without the complex semantic model
- The PF-transform not only makes the validation of such theory much simpler but also makes **direct** reasoning in the model viable
- Last point particularly useful in the case of incomplete theories (eg. separation logic)

Don't hesitate: join the PF-community and start "thinking big"

Addendum

ESC of "paths closed" invariant

$$id \times \Phi_{pc} \xleftarrow{f} (id \times \Phi_{pc}) \cdot \Phi_c \tag{130}$$

where (recall)

$$\begin{aligned} f &= FS\_DeleteFileDir\_System\ p \\ &= id \times (FS\_DeleteFileDir\_Tar\{p\}) \end{aligned}$$

PF-transformed $FS\_DeleteFileDir\_Tar$:

$$(FS\_DeleteFileDir\_Tar\ S)N = N \cdot \Phi_{\neg S} \tag{131}$$

Addendum

PF version of pc is

$$pc\ N \quad \triangleq \quad \underline{Directory} \cdot N \subseteq fileType \cdot attributes \cdot N \cdot dirName \tag{132}$$

cf. diagram

$$\begin{tikzcd}
FileHandler \arrow[r, "M"] & OpenFileDescriptor \arrow[dl, "path"'] & \\
Path \arrow[r, "N"'] & File \arrow[r, "attributes"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[r, "N"'] & File \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}$$

and the corresponding code in Alloy:

```
pred FileStoreInvariant[fs: FileStore]{
  (fs.tar).(File->Directory) in
  (dirName).(fs.tar).attributes.fileType}
```

Addendum

Picture of the whole model as far as data types are concerned,

$$
\begin{tikzcd}
\color{red}{OpenFileDescriptor} \arrow[d, "path"'] \arrow[dr, phantom, "\subseteq"] & \color{red}{FileHandler} \arrow[l, "\color{red}{M}"'] \arrow[d, "\top"] & \\
\color{red}{Path} \arrow[r, shift right, "\color{red}{N}"'] & \color{red}{File} \arrow[l, shift right, "N^{\circ}"'] \arrow[r, "attributes"] \arrow[d, phantom, "\rotatebox{90}{$\subseteq$}"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[r, "N"'] & File \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}
$$

where the two rectangules express datatype invariants.

Addendum

Strategy will be to ignore Φ_c for a moment and calculate the WP for f to preserve $id \times pc$; then we compare Φ_c with the pre-condition obtained. Thanks to (123), (130) becomes

$$\Phi_{pc} \xleftarrow{FS\_DeleteFileDir\_Tar\{p\}} \Phi_{pc} \qquad (133)$$

Below we generalize $\{p\}$ to any set of paths S and use abbreviations $ft := fileType \cdot attributes$ and $d := \underline{Directory}$:

$$\begin{array}{ll} & pc(FS\_DeleteFileDir\_Tar\ S\ N) \\ \Leftrightarrow & \qquad \{\ (131) \text{ and } (132)\ \} \\ & d \cdot N \cdot \Phi_{\neg S} \subseteq ft \cdot N \cdot \Phi_{\neg S} \cdot dirName \end{array}$$

Addendum

$$
\begin{array}{ll}
\Leftrightarrow & \qquad \{ \text{ shunting } \} \\
& d \cdot N \cdot \Phi_{\neg S} \cdot \mathit{dirName}^{\circ} \subseteq \mathit{ft} \cdot N \cdot \Phi_{\neg S} \\
\Leftrightarrow & \qquad \{ \text{ coreflexives versus conditions } \} \\
& d \cdot N \cdot \Phi_{\neg S} \cdot \mathit{dirName}^{\circ} \subseteq \mathit{ft} \cdot N \cap \top \cdot \Phi_{\neg S} \\
\Leftrightarrow & \qquad \{ \cap\text{-universal ; shunting } \} \\
& d \cdot N \cdot \Phi_{\neg S} \subseteq \mathit{ft} \cdot N \cdot \mathit{dirName} \;\wedge\; d \cdot N \cdot \Phi_{\neg S} \cdot \mathit{dirName}^{\circ} \subseteq \top \cdot \Phi_{\neg S} \\
\Leftrightarrow & \qquad \{ \text{ shunting ; } \top \text{ absorbs } d \; \} \\
& \underbrace{d \cdot N \cdot \Phi_{\neg S} \subseteq \mathit{ft} \cdot N \cdot \mathit{dirName}}_{\text{weaker than } pc(N)} \;\wedge\; \underbrace{N \cdot \Phi_{\neg S} \subseteq \top \cdot \Phi_{\neg S} \cdot \mathit{dirName}}_{\text{WP}}
\end{array}
$$

End of PF-calculation. Back to points:

Addendum

- Pointwise WP is as follows:

$$
\begin{array}{ll}
& \langle \forall\, q \;:\; q \in dom\ N :\; q \notin S \Rightarrow (dirName\ q) \notin S \rangle \\
\Leftrightarrow & \quad \{\ \text{logic}\ \} \\
& \langle \forall\, q \;:\; q \in dom\ N :\; (dirName\ q) \in S \Rightarrow q \in S \rangle
\end{array}
$$

that is: *if parent directory of existing path* q *is marked for deletion than so* q.

- For $S := \{p\}$:

$$
\begin{array}{ll}
& \langle \forall\, q \;:\; q \in dom\ N :\; (dirName\ q) = p \Rightarrow q = p \rangle \\
\Leftrightarrow & \quad \{\ \text{logic}\ \} \\
& \neg\langle \exists\, q \;:\; q \in dom\ N :\; (dirName\ q) = p \wedge q \neq p \rangle
\end{array}
$$

Addendum

Closing:

- c is indeed stronger than calculated WP
- In particular, it doesn't allow for *Root* deletion
- WP enables one to delete *Root* provided no other files exist in the FS.

  NB: *POSIX standard is ambiguous in this matter...*

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

R.C. Backhouse.
Fixed point calculus, 2000.
Summer School and Workshop on Algebraic and Coalgebraic Methods in the Mathematics of Program Construction, Lincoln College, Oxford, UK 10th to 14th April 2000.

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

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

.

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

P. Hoogendijk.
A Generic Theory of Data Types.
PhD thesis, University of Eindhoven, The Netherlands, 1997.

R. Joshi and G.J. Holzmann.
A mini challenge: build a verifiable filesystem.
Formal Asp. Comput., 19(2):269–272, 2007.

C. Necco, J.N. Oliveira, and J. Visser.
Extended static checking by strategic rewriting of pointfree relational expressions.
Technical Report FAST:07.01, CCTC Research Centre, University of Minho, 2007.

J.N. Oliveira.

Invariants as coreflexive bisimulations — in a coalgebraic setting, December 2006.
Presentation at the IFIP WG 2.1 #62 Meeting, Namur, Belgium. (Joint work with A. Silva and L.S. Barbosa. Slides available from the author's website).

J.N. Oliveira.
Pointfree foundations for (generic) lossless decomposition, 2007.
(Submitted).

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

J.N. Oliveira and C.J. Rodrigues.
Pointfree factorization of operation refinement.

In *FM'06*, volume 4085 of *LNCS*, pages 236–251.
Springer-Verlag, 2006.
.

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

Wang Shuling, L.S. Barbosa, and J.N. Oliveira.
A relational model for confined separation logic, Sep. 2007.
Submitted.

P.L. Wadler.
Theorems for free!
In *4th International Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359, London, Sep. 1989. ACM.