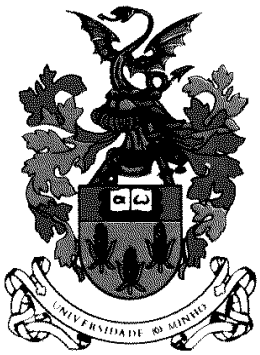
Fight Induction! Deduction is Forever.



— LOGCOMP Workshop —
6–December 1999
Lisboa
Portugal

J. Nuno Oliveira
Departamento de Informática
Universidade do Minho
Braga
Portugal

email: jno@di.uminho.pt
http://www.di.uminho.pt/~jno

About the Title

(which should really be **Research Topics in Morphism-based Data Refinement**) :

- *Memories of ADJ* [Gog89]:

  [...] *the main motivation for initiality was to replace tedious inductive arguments by diagram chases that exploit the existence of unique homomorphisms. [...] A key sentence [...] summarized this viewpoint with the slogan*

  Fight induction!

  The IBM censors insisted that this had to be removed. [...]

- Amir Pnueli's invited talk at **FM99**:

  Deduction is Forever.

Context

Real problems, formal models, calculi:

- Control systems, integro-differential equations, **Fourier**/ **Laplace** transforms
- Software systems, formal specifications, "**categorial** transform"

Morphisms in software development

A broad spectrum of research

- Software specification and calculation based on universal morphisms
- Rôle of universal properties

- Constructions replace inductive proofs (cf. algebraization of logic)
- Functorial constructions
  - Polymorphism, cf. natural transformations
  - Polytypism, cf. regular functors
  - Monads, cf. state, imperative constructs, I/O, etc
- Wide-spectrum interpretation of results:
  - Dualities, eg. algebra/coalgebra versus program/process
  - Change of category, eg
    - **Cpo** (strict functional programming, Bird-Meertens formalism)
    - **Rel** (pre-post calculi based on nondeterminism narrowing, logic programming)

However...

- Poor investment on **refinement** and **subtyping** (cf functional programming bias)
- Loose approach to algorithm **complexity** analysis
- Datatype **invariants** (and non-structural properties in general) not so easy to capture and reason about
- Non-initial types are very frequent (eg finite mappings, database models) but poorer theory-wise
- Temporal/modal logics: so far not a very significant investment

Scope

Inductive, regular types. For the “coinductive side of the coin” see [Bar00].

Background

Basic types

Product, coproduct, exponential, etc

Inductive types

$$\mu\mathsf{F} \underset{in}{\overset{out}{\cong}} \mathsf{F}\,\mu\mathsf{F}$$

Catamorphisms (*cf.* ADJ “unique homomorphisms”):

$$\begin{array}{ccc} \mu\mathsf{F} & \xleftarrow{in} & \mathsf{F}\,\mu\mathsf{F} \\ {\scriptstyle (\!|\alpha|\!)_{\mathsf{F}}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|\alpha|\!)_{\mathsf{F}}} \\ A & \xleftarrow[\alpha]{} & \mathsf{F}\,A \end{array}$$

Universal property

$$k = (\!|\alpha|\!) \;\Leftrightarrow\; k \cdot in = \alpha \cdot \mathsf{F}\,k$$

Corollaries of the universal property

Cata-reflection : $(\!|in|\!) = id_{\mu \mathsf{F}}$ (1)

$$\begin{array}{ccc} \mu\mathsf{F} & \xleftarrow{in} & \mathsf{F}\,\mu\mathsf{F} \\ {\scriptstyle (\!|in|\!)}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|in|\!)} \\ \mu\mathsf{F} & \xleftarrow[in]{} & \mathsf{F}\,\mu\mathsf{F} \end{array}$$

Cata-cancellation : $(\!|\alpha|\!) \cdot in = \alpha \cdot \mathsf{F}\,(\!|\alpha|\!)$ (2)

$$\begin{array}{ccc} \mu\mathsf{F} & \xleftarrow{in} & \mathsf{F}\,\mu\mathsf{F} \\ {\scriptstyle (\!|\alpha|\!)}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|\alpha|\!)} \\ A & \xleftarrow[\alpha]{} & \mathsf{F}\,A \end{array}$$

Cata-fusion : $f \cdot (\!|\alpha|\!) = (\!|\beta|\!)$ if $f \cdot \alpha = \beta \cdot \mathsf{F}\,f$ (3)

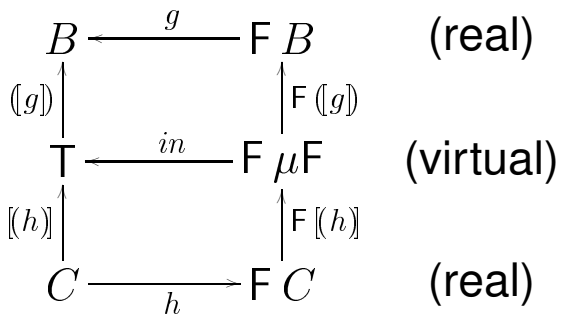
$$\begin{array}{ccc} \mu\mathsf{F} & \xleftarrow{in} & \mathsf{F}\,\mu\mathsf{F} \\ {\scriptstyle (\!|\alpha|\!)}\downarrow & & \downarrow{\scriptstyle \mathsf{F}(\!|\alpha|\!)} \\ A & \xleftarrow[\alpha]{} & \mathsf{F}\,A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,f} \\ B & \xleftarrow[\beta]{} & \mathsf{F}\,B \end{array}$$

Cata-absorption : $(\!|g|\!) \cdot \mathsf{T}\,f = (\!|g \cdot \mathsf{B}\,(f, id)|\!)$ (4)

$$\begin{array}{ccc} A & & \mathsf{T}\,A & \xleftarrow{in_A} & \mathsf{B}\,(A, \mathsf{T}\,A) \\ {\scriptstyle f}\downarrow & & {\scriptstyle \mathsf{T}\,f}\downarrow & & \downarrow{\scriptstyle \mathsf{B}\,(id, \mathsf{T}\,f)} \\ B & & \mathsf{T}\,B & \xleftarrow[in_B]{} \mathsf{B}\,(B, \mathsf{T}\,B) & \xleftarrow[\mathsf{B}\,(f,id)]{} \mathsf{B}\,(A, \mathsf{T}\,B) \\ & & {\scriptstyle (\!|g|\!)}\downarrow & \downarrow{\scriptstyle \mathsf{B}\,(id,(\!|g|\!))} & \downarrow{\scriptstyle \mathsf{B}\,(id,(\!|g|\!))} \\ & & C & \xleftarrow[g]{} \mathsf{B}\,(B, C) & \xleftarrow[\mathsf{B}\,(f,id)]{} \mathsf{B}\,(A, C) \end{array}$$

for $\mathsf{T}\,A \cong \mu X.\mathsf{B}\,(A, X)$.

(Hylo = Ana + Cata)-morphisms



A “gentle introduction” is attempted in [Oli99a, Oli99b].
Main reference: [BdM97].

A basis for a refinement taxonomy

Data rôle	Data structures	Refinement
I/O	Real	Data refinement
Intermediate	Virtual	Algorithmic refinement

Algorithmic refinement

This is in fact wider in scope:

- Change of virtual data-structure, or changing the algorithmic control:
  - binary-trees ($\mathsf{F}\,X = 1 + A \times X^2$) lead to $\mathcal{O}(log\ n)$ time complexity
  - left-lists ($\mathsf{F}\,X = 1 + X \times A$) lead to $\mathcal{O}(1)$ space complexity
  - (monoid) accumulations trim $\mathcal{O}(n^2)$ time complexity down to $\mathcal{O}(n)$ time complexity
  - vital rôle of exponentials!

- refinement by “sequential loop” inter-combination: fusion and absorption laws + deforestation (removal of intermediate data-structures)
- refinement by “parallel loop” inter-combination: mutual recursion elimination (Fokkinga’s law + “banana-split”)

Data refinement

The split-epic approach:

$$A \underset{f}{\overset{r}{\underset{\leq}{\overset{\leq}{}}}} B \quad \text{such that} \quad f \cdot r = id_A$$

f = abstraction; r = representation.

Right invertibility is preserved by arbitrary functors

$$A \underset{f}{\overset{r}{\underset{\leq}{\overset{\leq}{}}}} B \quad \Longrightarrow \quad \mathsf{F}\,A \underset{\mathsf{F}\,f}{\overset{\mathsf{F}\,r}{\underset{\leq}{\overset{\leq}{}}}} \mathsf{F}\,B$$

and by type-functors

$$\mathsf{F}\,X \underset{f}{\overset{r}{\underset{\leq}{\overset{\leq}{}}}} \mathsf{G}\,X \quad \Longrightarrow \quad \mu\mathsf{F} \underset{(\!|in_{\mathsf{G}} \cdot r|\!)_{\mathsf{F}}}{\overset{(\!|in_{\mathsf{G}} \cdot r|\!)_{\mathsf{F}}}{\underset{\leq}{\overset{\leq}{}}}} \mu\mathsf{G}$$

cf. diagram

$$\begin{array}{ccccc} \mu\mathsf{G} & & \mathsf{F}\,(\mu\mathsf{G}) & \xleftarrow{f} & \mathsf{G}\,(\mu\mathsf{G}) \\ {\scriptstyle (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}}} & & \downarrow{\scriptstyle \mathsf{G}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}}} \\ \mu\mathsf{F} & \xleftarrow[in_{\mathsf{F}}]{} & \mathsf{F}\,(\mu\mathsf{F}) & \xleftarrow[f]{} & \mathsf{G}\,(\mu\mathsf{F}) \end{array}$$

and constructive — *non inductive!* — proof (α is the unknown):

$$
\begin{array}{ll}
 & (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot (\!|\alpha|\!) = id \\
\Leftrightarrow & \quad \{ \text{ by F-cata-reflexion} \} \\
 & (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot (\!|\alpha|\!)_{\mathsf{F}} = (\!|in_{\mathsf{F}}|\!)_{\mathsf{F}} \\
\Leftarrow & \quad \{ \text{ by F-cata-fusion} \} \\
 & (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot \alpha = in_{\mathsf{F}} \cdot \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \\
\Leftrightarrow & \quad \{ \text{ decompose } \alpha = in_{\mathsf{G}} \cdot \alpha' \} \\
 & (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot in_{\mathsf{G}} \cdot \alpha' = in_{\mathsf{F}} \cdot \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \\
\Leftrightarrow & \quad \{ \text{ by G-cata-cancellation} \} \\
 & in_{\mathsf{F}} \cdot f \cdot \mathsf{G}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot \alpha' = in_{\mathsf{F}} \cdot \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \\
\Leftrightarrow & \quad \{ f \text{ is natural} \} \\
 & in_{\mathsf{F}} \cdot \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \cdot f \cdot \alpha' = in_{\mathsf{F}} \cdot \mathsf{F}\,(\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}} \\
\Leftarrow & \quad \{ f \cdot r = id \} \\
 & \alpha' = r
\end{array}
$$

So

$$\alpha \;=\; in_{\mathsf{G}} \cdot r$$

cf. diagram

$$
\begin{array}{ccc}
\mu\mathsf{F} & \xleftarrow{in_{\mathsf{F}}} & \mathsf{F}\,(\mu\mathsf{F}) \\
{\scriptstyle (\!|\alpha|\!)_{\mathsf{F}}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|\alpha|\!)_{\mathsf{F}}} \\
\mu\mathsf{G} & \xleftarrow[\alpha]{} & \mathsf{F}\,(\mu\mathsf{G}) \\
{\scriptstyle (\!|in_{\mathsf{F}} \cdot f|\!)_{\mathsf{G}}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(\!|\alpha|\!)_{\mathsf{G}}} \\
\mu\mathsf{F} & \xleftarrow[in_{\mathsf{F}}]{} & \mathsf{F}\,(\mu\mathsf{F})
\end{array}
$$

Invariants

Low-level invariants versus representation functions in data re-

finement:

$$A \underset{f}{\overset{r}{\underset{\leq}{\rightleftarrows}}} B \xrightarrow{\phi} 2$$

i.e.

$$f \cdot r = id$$
$$(r \cdot f)b = b \implies \phi\, b$$

as for instance in [dM99]:

$$A^\star \underset{flatten}{\overset{build}{\underset{\leq}{\rightleftarrows}}} \mathsf{BTree}\, A \xrightarrow{heap} 2$$

where

```
data BTree a = Null | Bin (BTree a) a (BTree a)
```

$flatten$ is in-order traversal, *etc.*

Particular interesting situation is

$$A \underset{f}{\overset{r}{\underset{\cong}{\rightleftarrows}}} B_\phi$$

where

$$\begin{array}{ccc} B_\phi & \xrightarrow{!_{B_\phi}} & 1 \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle \underline{\mathsf{TRUE}}} \\ B & \xrightarrow[\phi]{} & 2 \end{array} \qquad \phi(i\, b) = \mathsf{TRUE} \quad , \quad b \in B_\phi$$

Invariants and data "cleansing"

Problem is to identify exceptions ("dirty data") wrt. (large!) collections of "business rules" ϕ.

Reverse calculation approach: find f and r such as above and

run them as filters:

$$B \xleftarrow[r]{} A \xleftarrow[f]{} B$$

(clean data) (abstraction) (dirty data)

In the **KARMA** approach [NSO99], reverse specification A is automatically generated in ISO standard VDM-SL.

KARMA is based on a collection of rules, *e.g.*

$$A \rightharpoonup (B \times C) \underset{\bowtie}{\overset{\langle A \rightharpoonup \pi_1, A \rightharpoonup \pi_2 \rangle}{\cong}} ((A \rightharpoonup B) \times (A \rightharpoonup C))_{=\cdot(dom \times dom)}$$

in the relational model.

Think of the number of proof obligations (induction over finite mappings) discharged by calculation.
However: relationship between ϕ and f, r still has to be proved (we want to calculate it).

Sub-object pullbacks

One is able to show rules such as

$$\mathsf{T}\,(A_\phi) \;\cong\; (\mathsf{T}\,A)_{(\mathsf{T}\,\phi)}$$

and

$$A \underset{f}{\overset{r}{\leq}} B \quad \Longrightarrow \quad A_\phi \underset{f}{\overset{r}{\leq}} B_{\phi \cdot f}$$

but otherwise the pullback approach does not seem very practical.

Relations and monotypes

Drill everything down to relations, *e.g.*

$$build \;\subseteq\; heap? \cdot flatten^\circ$$

cf.

$$A^\star \underbrace{\overbrace{\leq}^{build}}_{flatten^\circ} \mathsf{BTree}\, A \xrightarrow[heap?]{} \mathsf{BTree}\, A$$

where f° is the converse of f and $p? \subseteq id_A$ is the coreflexive map induced by $2 \xleftarrow{p} A$.

However, need for power allegories:

> relations + inclusion, meet, converse and the power transpose $\mathcal{P}\, B \xleftarrow{\Lambda R} A$ of $B \xleftarrow{R} A$

+ relators (monotonic functors) + etc . . . !

The question is: point-free relation calculus? nondeterministic mappings?

Nested datatypes

These are based on non-regular functors, *e.g.* [BM98]:

```
data Nest a = NilN | ConsN (a, Nest (a,a) )
```

It so happens that

$$\mathsf{Nest}\, a \underbrace{\overbrace{\leq}^{r}}_{f} (\mathsf{Tree}\, a)^\star_\phi$$

where

```
data Tree a = Val a | Bin (Tree a) (Tree a)
```

and

$$\phi\, l \;=\; \forall 1 \leq i \leq \mathsf{length}\, l : i = depth(l\, i) \wedge balanced(l\, i)$$

So, ϕ could be captured (implicitly) by using $\mathsf{Nest}\, a$ as intermediate data-structure.

Very promising, but a lot of work remains to be done.

Tabular implementation of recursion

"Recursion removal" theorem:

$$\mu\mathsf{F} \underset{f}{\overset{r}{\underset{\leq}{\rightleftarrows}}} ((K \rightharpoonup \mathsf{F}\, K) \times K)_{\phi}$$

Since

$$(\mu\mathsf{F})^K \xleftarrow{\overline{f}} (K \rightharpoonup \mathsf{F}\, K)$$

one may think of modelling the "heap" by a (partial) coalgebra:

$$\begin{array}{ccc} \mu\mathsf{F} & \xleftarrow{in_{\mathsf{F}}} & \mathsf{F}\,(\mu\mathsf{F}) \\ {\scriptstyle \overline{f}=[\![\sigma]\!]}\big\downarrow & & \big\downarrow{\scriptstyle \mathsf{F}\,\overline{f}} \\ K & \xrightarrow[\sigma]{} & \mathsf{F}\,K \end{array}$$

Example, for HASKELL right-lists, *i.e.* $\mathsf{F}\,X = 1 + A \times X$:

```
import Mpi
import RList

f = uncurry anaRList

sigma 1 = i2("a",3)
sigma 2 = i2("b",4)
sigma 3 = i2("a",2)
sigma 4 = i1(())
```

Then

$$f(\sigma, 1) = [a, a, b]$$

but

$$f(\sigma, 5) = \bot$$

Pointer definedness and well-foundedness can be guaranteed by ϕ as in [Oli92]. However, the pointwise proof by Jourdan [Jou92] (induction on the structure of polynomial F) is several A4 pages long.

Challenge: do it by pointfree calculation.

Other challenges: fractal types

Fractal types, which generalize hash-tables [Oli98], should satisfy

$$\mathsf{T}\,(n \times A) \underset{f}{\overset{r}{\cong}} (\mathsf{T}\,A)^n$$

Examples:

- Exponentials: $\mathsf{T}\,X = B^X$ is fractal, recall "currying".
- Partial mappings: $\mathsf{T}\,X = X \rightharpoonup B$ is fractal since

$$(n \times A) \rightharpoonup B \;\cong\; (A \rightharpoonup B)^n$$

However, is $\mathsf{T}\,X = X^*$ fractal?

$$(n \times A)^\star \underset{?}{\overset{r}{\cong}} (A^\star)^n$$

Last but not least

Need for complexity analysis based on morphism calculation.

Algebra of cost functions? Eg.

$$\begin{aligned} cost(\llbracket \mathcal{O}(1), \mathcal{O}(n) \rrbracket)_{\mathsf{RList}} &= \mathcal{O}(n^2) \\ \ldots &= \ldots \end{aligned}$$

Final Remarks

- Need for strong support for refinement by calculation
- Good prospect for applications: advanced compiling techniques (eg fusion, deforestation), specification language compilers, reverse calculation, formal re-engineering methods, legacy systems, data quality, etc

References

[Bar00] L. S. Barbosa. *Towards a Calculus of Process Refinement*. University of Minho, 2000. Ph. D. thesis (forthcoming).

[BdM97] R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997. C. A. R. Hoare, series editor.

[BM98] R. Bird and L. Meertens. Nested datatypes. In J. Jeuring, editor, *MPC'98: Mathematics of Program Construction*, number 1422 in Lecture Notes in Computer Science, pages 52–67. Springer, 1998.

[dM99] O. de Moor. *Pointwise* relational program derivation. Technical report, Oxford PRG, June 1999.

[Gog89] J. A. Goguen. Memories of ADJ. *Bulletin of the European Assoc. for Theoretical Computer Science*, (36):96–102, Oct. 1989.

[Jou92] I. S. Jourdan. Reificação de tipos abstractos de dados: Uma abordagem matemática. Master's thesis, University of Coimbra, 1992. (In Portuguese).

[NSO99] F. L. Neves, J. C. Silva, and J. N. Oliveira. *Converting Informal Meta-data to VDM-SL: A Reverse Calculation Approach* . In *VDM in Practice! A Workshop co-located with FM'99: The World Congress on Formal Methods, Toulouse, France, 20-21 September*, 1999.

[Oli92] J. N. Oliveira. *Software Reification using the SETS Calculus* . In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).

[Oli98] J. N. Oliveira. *'Fractal' Types: an Attempt to Generalize Hash Table Calculation*. In *Workshop on Generic Programming (WGP'98), Marstrand, Sweden*, June 1998.

[Oli99a] J.N. Oliveira. An introduction to pointfree programming, 1999. 37p.

[Oli99b] J.N. Oliveira. Recursion in the pointfree style, 1999. 33p.