The Essence of Reynolds

3. State and Abstraction

Uday S. Reddy[1]

[1]University of Birmingham

POPL, 2014

John C. Reynolds, 1935-2013

Emmy Noether, 1882-1935

- According to Mac Lane, she “emphasized the importance of homomorphisms.”
- Within 10 years of her passing, category theory was born, taking “homomorphisms” as the primary elements, and formulating naturality as the preservation of homomorphisms.

- **John Reynolds** emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.
- Can we hope for a new theory within 10 years of his passing?
- Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?
- Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?
- Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”?

Logical relations

- Logical relations are relations compatible with structure, just like homomorphisms are functions preserving structure. See Hermida, Reddy and Robinson [2014].
- Other names for logical relations in the literature:
  - Regular relation, Homogeneous relation, Compatible relation (algebra)
  - Congruence relation (algebra - for *equivalence* relations)
  - Covering relation (automata theory)
  - Simulation relation, Bisimulation relation (Milner, CCS & pi-calculus)
  - Refinement mapping (Abadi & Lamport, Dist. systems)
- Relational parametricity asks for "parametrically polymorphic" functions to preserve all logical relations.

$$
\begin{array}{ccccc}
A & \qquad & F(A) & \xrightarrow{t_A} & G(A) \\
R \updownarrow & & F(R) \updownarrow & & \updownarrow G(R) \\
A' & & F(A') & \xrightarrow{t_{A'}} & G(A')
\end{array}
$$

Peter Freyd

I took this affront to category theory as a challenge.
There were several years that I often found myself thinking — and then saying out loud when lecturing —
that if we were to work very, very hard, we might catch up to where John Reynolds was years ago.[2]
— Core algebra revisited, 2007

2 I must record John's words when he attended one such lecture:

"You too? I've long been trying to catch up to where I used to be".

Relational parametricity

- 1974: Towards a theory of type structure
  - Polymorphic lambda calculus
  - Representation independence theorem (using Galois connections for complete lattices).
- 1983: Types, abstraction and parametric polymorphism
  - Relational parametricity
  - Abstraction theorem (generalizing “representation theorem”)
  - Uses relations instead of Galois connections

Why relations?

- Reynolds was a co-inventor of “logical relations” (1974), along with Plotkin (1973), Milne (1974).
- The crux of logical relations is this formula for properties of functions $f : A \to B$:

$$(P \to Q)(f) \iff \big(\forall x.\, P(x) \implies Q(f(x))\big)$$

- Or, for binary relations:

$$(R \to S)(f, f') \iff \big(\forall x, x'.\, R(x, x') \implies S(f(x), f'(x'))\big)$$

- Most people think of this as a proof technique to get induction to work.

Gordon Plotkin

- Plotkin's [1980] use of logical relations was much deeper.

> *"Because of the "logical" nature of λ-definable elements, they should be invariant under the permutations of D."*
>
> — *Lambda definability in the full type hierarchy*

- Elaborating:

Other types	**network connections**
"Mathematics"	**integer**, **real**
"Logic"	$\times$, $\to$

The logical constructions of λ-calculus can't "see" the mathematical types that lie above.

- Plotkin's "logical" = Reynolds's "parametric" ?

Abstraction

- Logical relations were also invented in automata theory [Ginzburg and Yoeli, 1965, Eilenberg, 1976].
- Milner [1969] rechristened them "simulation relations" and applied them to programming theory. Later bisimulations.
- Hoare [1972] applied the idea to data representations (abstract data types).
- Reynolds [1972-1981] used all these ideas in Data representation structuring (Ch. 5 of *Craft of Programming*).
- Mitchell and Plotkin [1982] made the connection between abstract types and existential types.

Reynolds's idea

- In putting all these ideas together, Reynolds made his characteristic giant leap:

  > *"The way of out of this impasse is to generalize homomorphisms from functions to relations."*
  >
  > — *Types, Abstraction and Parametric Polymorphism, 1983*.

- Homomorphisms represent the very foundation of the 20'th century mathematics!
- But Reynolds says, they only work for first-order types.
- We must generalize homomorphisms to logical relations to handle higher-order types.

Information hiding (Abstraction)

- In automata, process calculi and abtract types, computations are black boxes with hidden state:

  $$M : \exists Q.\, F(Q)$$

  This is “global” information hiding.
- Parametric polymorphism gives you “local” information hiding.

  $$t : \forall Q.\, F(Q) \to G(Q)$$

What next?

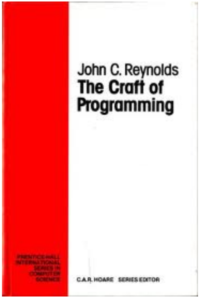
- Information hiding, a Computer Science idea, is also at the heart of mathematics.
- Reynolds parametricity gives us a mathematical theory of information hiding, which should have a wide range of applications.
- Computer Science has an opportunity to cause a “disruptive change” to 21st century Mathematics and, throught it, perhaps all of science.
- Some beginnings:
  - Hermida, Reddy, Robinson [2014].
  - Atkey [2014] — in this POPL.

Section 2

State

The Craft of Programming

- Reynolds was a superior imperative programmer (Algol 60, Algol W, Algol 68, assembly).
- Between 1972-1981, he taught a graduate course on programming at Syracuse, developing the work published as The Craft of Programming.



The Craft of Programming (contd)

- Contains a wealth of information about:
  - what imperative programming means,
  - how to develop imperative programs rigorously,
  - the type structure of imperative programs,
  - reasoning principles (both *practical*, e.g., arrays, and *abstract*, e.g., Specification Logic),
  - how to reason about data abstraction (or information hiding).
- Separation Logic, 2000, may be seen as a continuation of this body of work.

The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
  - 1978: Syntactic Control of Interference.
  - 1979: Reasoning about arrays.
  - 1980: Using Category Theory to Design Implicit Conversions and Generic Operators.
  - 1981: The Essence of Algol.
  - 1982: The Idealized Algol and its Specification Logic.
  - 1983: Types, Abstraction and Parametric Polymorphism.
  - 1984: Polymorphism is not set-theoretic.
- What do we see here?
  - The ideas of state, types, data abstraction, polymorphism are all interconnected in Reynolds's mind.
  - **State** is the key.

State and abstraction

Two major insights:

- Procedures of Algol 60 = typed lambda calculus.
  Recall: Call-by-name!
- *A Polymorphic Model of Algol* [Notes dated 1975]:
  Algol types are "type constructors," parameterized by state types. .

$$\begin{array}{rcl} \mathbf{com}[S] & = & S \rightharpoonup S \\ (\theta_1 \to \theta_2)[S] & = & \forall S'.\, \theta_1[S \times S'] \to \theta_2[S \times S'] \end{array}$$

- Whereas pure functional programming lives in classical set theory, imperative programmign works in intuitionistic set theory (Kripke-style, presheaf model).
- Contrast with Strachey's denotational semantics, which essentially tries to reduce imperative programming to functional programming, i.e., classical set theory.

Interpreting Algol types

- The correspondence between Algol and functional programs:

Algol (intuitionistic)	classical functional
types	type constructors (functors/relators)
terms	polymorphic functions (natural/parametric transformations)

- So, first-order Algol programs become polymorphic higher-order functional programs.
- Parametricity implies: $[\mathbf{com} \to \mathbf{com}] \cong \mathbb{N}$. (a form of Church numerals).
- Hoare's proof rule for while loops is a free theorem.

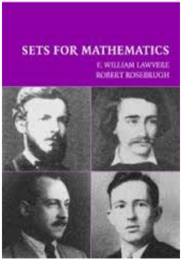
Intuitionistic set theory (Kripke models)

- An "intuitionistic" set (in the sense of Kripke and Lawvere) is a set parameterized by some context, which we call a "world."

$$A(X) = \ldots$$

Lawvere (Sets for Mathematics, 2003) also calls them "variable sets." Normal sets are called "constant sets."

- In the Algol case, "worlds" are store shapes.



- Functions: $(A \Rightarrow B)(X) = \forall_{Y \geq X} A(Y) \rightarrow B(Y)$
- Intuitively: a "function" of type $A \Rightarrow B$ at world X can work at every future world Y, accepting arguments of type A at world Y and giving results of type B at world Y.

Modularity in State

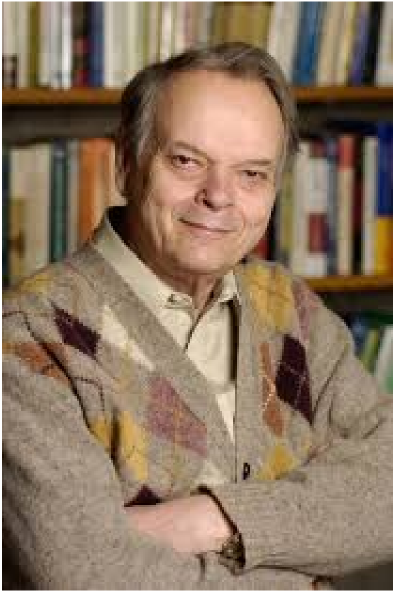
- Since state is an implicit type parameter, Reynolds thought about further information hiding aspects to capture modularity in state.
- *Syntactic Control of Interference* [1978]:

  $$(\theta_1 \multimap\!\star\; \theta_2)[S] \quad = \quad \forall_{S'}\ \theta_1[S'] \to \theta_2[S \times S']$$

  Procedure and its argument should not "interfere" (depend on separate portions of storage).
- Corresponding non-interfering product [O'Hearn et al.]:

  $$(\theta_1 \star \theta_2)[S] \quad = \quad \exists_{S_1, S_2 \mid S_1 \times S_2 \leq S}\ \theta_1[S_1] \times \theta_2[S_2]$$

- Separation logic [2000-2013] reinterprets these ideas for predicates instead of types.
- A fitting culmination of a lifelong quest for understanding the deepest underpinnings of programming languages!



John C. Reynolds, 1935-2013