The inner life of programming concepts

Tomas Petricek, The Alan Turing Institute
tomasp.net | tomas@tomasp.net | @tomaspetricek
Motivation

What is this talk about?
Programming concepts

• Where do they come from?
• How do they evolve?
• How are they used in practice?
• What is their nature?
Programming concepts
Multi-level nature of programming concepts
1. Metaphorical common sense

2. Formal models

3. Computer implementation
<table>
<thead>
<tr>
<th>Types</th>
<th>Monads</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>category or kind</td>
<td>box or sequencing</td>
<td>reusable block</td>
</tr>
<tr>
<td>sets or relations</td>
<td>category theory</td>
<td>math function</td>
</tr>
<tr>
<td>ML or Java types</td>
<td>Haskell or LINQ</td>
<td>FORTRAN or ML</td>
</tr>
</tbody>
</table>
Multi-level nature of concepts

ALL HAVE THREE LEVELS

We always think, prove and implement!

ONE LEVEL SOMETIMES DOMINATES

Common sense types, formal monads, implementation functions
Programming concepts as entities
Learning from mathematical and scientific entities
Realism and anti-realism debate

MORE FLEXIBLE THAN ELECTRONS

Paper on gravity does not change how apples fall

NOT JUST ABSTRACTION

Types across languages don't share the abstract
Experimentalist view

CAUSING EFFECTS
Types are used to provide auto-complete

SCIENTIFIC PROGRESS
We accumulate what we can implement
Learning from mathematical entities

HOW PROGRAMMING CONCEPTS EVOLVE

Proofs and refutations (Imre Lakatos)

WHERE PROGRAMMING CONCEPTS COME FROM

Conceptual metaphors (George Lakoff & Rafael Núñez)
Monads as mathematical entities

Variations on proofs and refutations
MATHEMATICAL LAYER

Monad is a monoid in the category of endofunctors

IMPLEMENTATION LAYER

Data type $M^a$ satisfying monad laws with operations:

- `return` of type $a \to M^a$
- `>>=` of type $(a \to M^b) \to (M^a \to M^b)$
**Metaphorical layer:** Monads as boxes

- `return` wraps thing in a box

- `>>>=` applies operation on all things in a box
Metaphorical layer: Monads as computations

Plumbing for composing computations with side-effects
Metaphorical layer: Monads as computations

Plumbing for composing computations with side-effects

partial functions

- $\bullet \rightarrow \star$
- $\bullet \rightarrow \star$
- $\bullet \rightarrow \star$
Metaphorical layer: Monads as computations

Plumbing for composing computations with side-effects

partial functions

monadic bind

- [Diagram of partial functions and monadic bind]
Metaphorical layer: Monads as computations

Plumbing for composing computations with side-effects
Monad at multiple levels

FORMAL LEVEL
Pre-sheaves in category theory

METAPHORICAL LAYER
Containers like lists and sequencing of effects

IMPLEMENTATION LAYER
Artifacts such as do notation and LINQ
Shifts and adaptations

Motivation at formal level
Monads are logic for reasoning about effects

Used differently for implementation
Language abstraction for encoding effects

Shift at implementation level
Abstraction and notation for effects

Causes adaptation at formal level
Algebraic reasoning about syntactic structures
Types, functions and more

Variations on proofs and refutations
Types in programming languages

FORMAL LAYER
Types as sets, types as relations, types as proofs

IMPLEMENTATION LAYER
Types for error checking, assisting developers

METAPHORICAL LAYER
Category of a value, property specification
How the concept of a type evolves

Rebirth at the implementation layer
No clue that "type" from logic had role in early Algol

Modelled at the formal layer
Type as a set of things (except for pointers)

Shifts at the application layer
Types used for effect tracking, tooling, proving

Adaptation at the formal layer
Types as relations, types as proofs
Functions in programming languages

**IMPLEMENTATION LAYER**
Gluing tapes, compiling sub-routines

**FORMAL LAYER**
Mapping from inputs to outputs

**METAPHORICAL LAYER**
Mathematical lookup tables, math functions
How the concept of a function evolves

- Formal and implementation appearance
  Mathematics vs. gluing sequences of instructions
- Getting closer at implementation layer
  Functions and procedures in Pascal
- Implementation and metaphorical shifts
  Sending a message to an object in Smalltalk
- Adaptation at the formal layer
  Unit-returning function with side-effects
Summary

Why understanding programming concepts matters
Computer science method questions

- Metaphors appears only in textbooks
- How to discuss conceptual metaphors
- What to study about implementation level
History and philosophy questions

- Origins of programming concepts
- Link between formal and implementation
- Proofs and refutations like developments
Shifts and adaptations?

- **Shift** at one layer, followed by **adaptation** at another
- **Joining** two concepts, **splitting** one concept
- **Reversal** of a shift caused by another layer
The inner life of programming concepts

Multi-level nature
Conceptual, formal, implementation

Useful philosophy of science perspective
Where programs come from, scientific progress

Variations on proofs and refutations
Evolve across all three levels

tomasp.net | tomas@tomasp.net | @tomaspetricek
BACKUP SLIDES
Concepts as technical artifacts

DUAL NATURE OF TECHNICAL ARTIFACTS
Function or specification vs. physical implementation

FUNCTION OF PROGRAMMING CONCEPTS
Changes and evolves and stretched by implementations
Physical reality

Hard to cover all philosophy of science positions!

Scientific entities
There is some independent physical reality

Programming concepts
The physical is interlinked with the theoretical