Preparing Programmers for Quantum

J.N. Oliveira
Research context

- Bridging U.Minho / INESC TEC / INL (Braga, Portugal)
- Targeting reversible and quantum programming from a formal method’s perspective.
Happy blend of diverse bodies of knowledge:

- **Philosophy**
  - Formal Logic
  - Semiconductor Electronics

- **Maths**
  - Automata Calculus
  - Grammars Languages

- **Physics***
- **Linguistics***

(Source: Manchester University, UK)
Maths dreamed of it...

1936

A. Turing — abstract notion of what we now call a programmable computer — known as the Turing machine.

1936

A. Church — λ-calculus, the basis of functional programming.
Physics made it happen...

Vacuum tubes, triodes (1912)

(Credits: https://en.wikipedia.org/wiki/Triode)
Physics made it happen...

Transistors (1948)
... but soon abstraction was needed
... first graphical, then formal

**Graphical**

![NAND gate diagram]

**Formal logic**

\[
nand : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\} \\
nand (x, y) = \neg (x \land y)
\]
50 years later...

1985

David Deutsch (U. Oxford) describes the first universal quantum computer.

Nowadays

Physics making it happen, again...

History repeating itself?

Source: IBM Q Experience website
The big picture

1950s — “L’enfant terrible” is born (Adapted from 2015)
Crisis (1960s)

1st NATO Conference on Software Engineering, Garmisch, Oct. 1968
Phrase **software engineering** seems to date from the Garmisch NATO conference in 1968:

_In late 1967 the Study Group recommended the holding of a working conference on Software Engineering._

_The phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering._

How **scientific** is SE today?
Not everybody looks happy...

Where is the Science in Computer Science?

“... we have a responsibility to pursue the science in computer science. We must develop better tools and much deeper understanding of the systems we invent and a far greater ability to make predictions about the behavior of these complex, connected, and interacting systems.”.

Software as a problem

Software

\[
\begin{align*}
\text{Process} & \quad \checkmark \\
\text{Product} & \quad \text{?}
\end{align*}
\]
“L’enfant terrible”

Unlike **hardware, software** not governed by the laws of **physics**:

- does not weight, does not smell,
- it is chemically neutral ...

**Anthony Oettinger (ACM President, 1967):**

"(...) the scientific, rigorous component of computing, is more like mathematics than it is like physics".

A. Oettinger (1929-)
Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is
Old concerns

Still Oettinger (already in 1967):

"It is a matter of complexity. Once you start putting thousands of these instructions together you create a monster which is unintelligible to anyone save its creator and, most of the time, unfortunately even to the creator."
Old concerns

Still Backus (1978):

"Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects (...) cause them to be both fat and weak: (...) their inability to effectively use powerful combining forms for building programs from existing ones, and their lack of useful mathematical properties for reasoning about programs."
Interestingly...

Backus (1978) predicted the age of **MapReduce**, written

\[(/g) \cdot (\alpha f)\]

in his **FP combinator** notation.
Meanwhile...
What should have happened?

**IS ABSTRACTION THE KEY TO COMPUTING?**

Why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot? Is it possible to improve these skills through education and training? Critical to these questions is the notion of abstraction.

__________________________
By JEFF KRAMER

(Commun. ACM, 50:4, pages 37–42, April 2007)
Abstraction

Quoting Jeff Kramer:

**Abstraction** is widely used in other disciplines such as art and music. (...) Henri Matisse manages to clearly represent the essence of his subject (...) using only simple lines or cutouts. His representation removes all detail yet conveys much.
Abstraction

The famous “abstract map” of London’s Underground (1939).

The art of removing unnecessary detail.
Abstraction

Rather than “WYSIWYG”, we want

Expressiveness = \frac{\text{What you get}}{\text{What you write}}

Expressiveness versus productivity:

“(…) Readers of this book will enjoy a rare opportunity to learn how to write less in order to say more, without ambiguity. In short, to learn how to be productive”.

(endorseing D. Jackson’s Software Abstractions, MIT Press, 2011)
Quantum abstraction

- Programmers to be challenged even further by abstractions in quantum programming
- Abstract from the physical layer! (quantum physics difficult and counter-intuitive)
- Get it right from the very beginning! (debugging nearly impossible)
- Avoid reading the state — measuring interferes with the quantum effect!
Quantum computing literature
Quantum computing literature
Quantum computing literature

- Degree of sophistication varies
- Emphasis on the need to change one’s mindset (as happened to physicists)
- Some pedagogical effort
- What kind of abstractions are involved?
Why do functions always come first?

Example 3.2 Some process theories we will encounter are:

- **functions** (types = sets)
- **relations** (types = sets, again)
- **linear maps** (types = vector spaces, or Hilbert spaces)
- **classical processes** (types = classical systems)
- **quantum processes** (types = quantum and classical systems)

(Excerpt from chapter 3 of Picturing quantum processes by Coecke & Kissinger)
Quantum programs (circuits)

Quantum circuit generated using the Quipper tool:

Functional abstraction:
Similar abstractions in Neural Networks

(RNN = accumulating maps)

(Source: Neural Networks, Types, and Functional Programming by C. Olah, 2015)
A functional quantum program

\[
\lll (a, b) \rightarrow \text{Vec}(c, b)) \rightarrow ([a], b) \rightarrow \text{Vec}([c], b)
\]

\[
\lll [] \rightarrow \text{return} ([], b)
\]

\[
\lll h : t, b) = \text{do} \{
\]
\[
(t', b') \leftarrow \lll (t, b);
\]
\[
(h'', b'') \leftarrow f(h, b');
\]
\[
\text{return} (h'' : t', b'')
\]

It controls \textbf{qubit} \textit{b} according to a list of classical bits using the \textbf{quantum} operator \textit{f} (parameter).

How does it compile?
Current experiments

Tool-chain:

Example of quantum circuit generated from the given program:

(With thanks to: Ana Neri, Afonso Rodrigues, Rui S. Barbosa)
FP on the quantum way

- **Quantum programs** generalize (reversible) *functions*
- Quantum programs much closer to **functional programs** (FP) than to **imperative** ones.

Meanwhile

- functional **flavour** is spreading across languages (F#, Swift, Java 8, Python, ...).
Too late

Question:

Should FP be taught at universities as first language?

Answer:

▶ Yes, it should (and this is the case in several places)

However:

▶ That’s simply... too late!

J. McCarthy (1927-2011)
From around age 12 to adulthood, individuals indicate an ability to think abstractly, systematically, and hypothetically, and to use symbols related to abstract concepts. This is the crucial stage at which individuals are capable of thinking abstractly and scientifically.

(Kramer, 2007)

VnC “junior university” courses, University of Minho, every July — introducing ISCED\(^1\) level 3 students (15 to 18) to programming.

\(^1\)International Standard Classification of Education.

At our VnC module we have 10 years of experience in teaching FP to ISCED level 3 students.

As a rule, students like the course.

(Even those who have programmed before.)
Formal operational stage

Alongside with **maths**, **physics** and the other subjects, **middle school** students should study basic **computer science** and **programming**.

**FP** blends very nicely with **maths** and **physics**.

Such has been our experience at **VnC** (2007-17).
Imagine...

You may say ___ I’m a dream-er___

But I’m not ___ the only one___

(Lennon, 1971)
Our department has long been an advocate of the functional-first school of programming and has been teaching Haskell as a first language (...) for 20 years. (...) We have been using game programming to keep students motivated (...) We summarise (...) our experience [in developing] a model for comprehensive and interactive functional game programming assignments (...)

“Computing & Schools” (CAS) trend

Computing as a mandatory subject on the EU (2016)
• **2014, UK**: covering Primary and Secondary school
• **2016, France**: covering 2nd, 3rd and 4th cycles
• **2016-18, Finland**: covering Primary and Secondary school
• **2016-17, Poland, Malta and Croatia**: covering Primary and Secondary school
• **2017, Denmark**: covering Secondary school
• **Spain, Belgium and Germany**: introduced at a regional level

(Credits: N.F. Rodrigues)
2018's Software Engineering Talent Shortage— It’s quality, not just quantity

Forrester projects that firms will pay 20% above market for quality engineering talent in 2018
Future jobs...

What is a software engineer?

If you know a programming language, then are you an engineer? No. Knowing a language does not make you an engineer. The same as knowing how to speak elementary Spanish does not automatically make you a good Spanish teacher.

In-demand software engineers are problem solvers, not coders.

(https://hackernoon.com)
Thanks