Christopher Strachey

and the

Programming Research Group
Fox (Computer Journal, 1961):

It is certainly true that mathematicians ... should know what computing is about, and something about the nature of programming of machines. On the other hand it would be a degenerate step to replace, say, the theory of convergence, or even a more abstract topic in the mathematics syllabus, by a course designed to produce a breed of professional programmers.

...

This might seem obvious, but there seems to be a great need to labour it. I have the impression that some people think that programming ... is not only more important than mathematics but can actually replace mathematics, that, in a sense it is mathematics.

... the idea must be resisted strenuously that an ability to code a machine is synonymous with the mathematical ability necessary to make the best use of this remarkable equipment.
Response

Gill and Strachey:

Does not practical work on such a scale inevitably call for theoretical work on studies of common principles -- in this case the theories of programming languages, algorithms, compiling processes, list structures, recursive functions etc. ...? Are these not mathematics?

[Computers] are destined to play a part so basic and revolutionary that a correct appraisal of them is essential to our survival as an important nation. If this is so then the professional mathematicians must play a leading part ... as true mathematicians. They must do for computer programs what the famous mathematicians of the past have done for real and complex numbers.
Comeback

Fox:

Any undergraduate teaching of topics in “programming languages, compiling processes, list structures, recursive functions, etc.” must be associated, in black and white, with a syllabus. The Faculty is disinclined to alter a syllabus, and no amount of Churchillian rhetoric about ... “our survival as an important nation”, “a leading part”, “true mathematicians”, ... “famous mathematicians of the past” and so on, will bring about this change.

But:

[Gill and Strachey] could also perform a valuable service by entering temporarily the teaching world. ... I would be very happy to organize in 1963 a Summer School on "Non-numerical algorithms", "Artificial Intelligence", or whatever title is thought desirable ... . A concrete approach of this kind is, I suggest, the only way of getting their ideas across.
The Result

- Summer School happens in 1963


- Fox obtains SRC Research Grant to set up a research group at Oxford

- PRG starts in April 1966 with Strachey as leader
  - Strachey, initially paid from the SRC grant, is “taken over” by the University as an *ad hominem* Reader in August 1967 (at which time the grant is extended till 1972).
Two Seminal Books

  - Proceedings of conference in Vienna (1964)
    - Semantics by symbol manipulation (→ Algol 68)
    - Syntax, Chomsky languages etc.
    - The CUCH (Bohm)
    - λ-calculus (Landin)
    - Semantics by functions (Strachey)
Strachey’s Objective:

“It has long been my personal view that the separation of practical and theoretical work is artificial and injurious.

Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work.

Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing.

One of the central aims of the Programming Research Group as a teaching and research group has been to set up an atmosphere in which this separation cannot happen.”
Initial Work

- **“Theory”**
  - Tidying up CPL Papers (several CPL meetings)
  - *Fundamental Concepts in Programming Languages*
  - Compound Data Structures (Park)
  - Continuing work on program schemata (Park)
  - “Mathematical Semantics” – for first Diploma students

- **“Practice”**
  - (Attempts to move CPL from Titan to Atlas)
  - Experiments on KDF9
    - Streams
  - Transfer of other work to KDF9
    - GPM
    - BCPL

- **Visitors**
  - Doug McIlroy works on coroutines (invents pipes)

**McIlroy**: “I went to Oxford for a year, solely so I could imbibe denotational semantics from the source.”

**Aho**: “Doug McIlroy, though, I think is probably the author of translation...of pipes. That he had written, I think, this unpublished paper when he [was] at Oxford back in the '60s....You should read this paper because it's UNIX pipes. One of the interesting things about Doug is that he has had these great, seminal ideas which not everyone knows about. And whether his standards are so high that he doesn't publish them...or what? But it's remarkable...”
<table>
<thead>
<tr>
<th>Year</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1969 | Event: System for 3D data written in Depl. 0 and compiled by NIDA. Iota virtual machine code.  
      | Event: 3D graphics. System working with 48 hours.  
      | Event: Meeting at WGB 2 in 1970. AS reveals DIS.  
      | Event: Software for 3D graphics and computer.  
      | Event: 80,000 TCU for mainframe migration.  
      | Event: Settled at Oxford.  
      | Event: Development of Theory of Reflection.  
      | Event: Start of software for mainframe.  
      | Event: TM 1, PRG 1, DPMI: 12x4, 2x3x3 |
| 1970 | Event: Technical paper PRG 1  
      | Event: Work in visual machines.  
      | Event: TM 2, TM 4, DPMI: 1x4, 0x2x3, 12x3, 3x3x3 |
| 1971 | Event: TD 1, TM 5, DPMI: 1x4, 0x2x3, 3x3x3  
      | Event: Technical paper PRG 2, 3  
      | Event: Report on PRG by P. H. Brown (U. S. R.)  
      | Event: Technical paper PRG 7, 14 |

Table entries include:  
- TM: Time Machine,  
- PPM: Parallel Processing Machine,  
- DPMI: Data Processing Machine Interface.
1969

- (1968) JH Morris’s thesis (MIT): proves minimality of $Y$ ($Y$ is “worst possible” operator)

- PRG gets its own computer
  - (32K store; paper tape; no disc)
  - Prepare system on KDF9 in Jan, Feb
  - One visit to Hemel Hempstead
  - Machine arrives in March; system working within 48hrs.

- Dana Scott visits in Michaelmas Term
  - Strachey meets Scott at WG2.2 in Vienna (April)
  - Long weekly seminars in Michaelmas Term
    1) “Type-theoretic alternative to CUCH, ISWIM, OWHY”
    2) More explicit representation and derepresentation
    3) Inverse limit construction for reflexive domain

- PRG plans to publish Technical Monographs
Morris’s Thesis Proof

- **Definition:** $A = B : A \text{ cnv}_B B$.

- **Definition:** $A \supset B :$ For any $E$, whenever $E[B]$ has a normal form, $E[A] = E[B]$.

- **Theorem:** If $A \supset FA$, then $A \supset YF$.

- **Corollary:** If $A = FA$, then $A \supset YF$. 
1969

- (1968) JH Morris’s thesis (MIT): proves minimality of $Y$ ($Y$ is “worst possible” operator)

- PRG gets its own computer
  - (32K store; paper tape; no disc)
  - Prepare system on KDF9 in Jan, Feb
  - One visit to Hemel Hempstead
  - Machine arrives in March; system working within 48hrs.

- Dana Scott visits in Michaelmas Term
  - Strachey meets Scott at WG2.2 in Vienna (April)
  - Long weekly seminars in Michaelmas Term
    1) “Type-theoretic alternative to CUCH, ISWIM, OWHY”
    2) More explicit representation and derepresentation
    3) Inverse limit construction for reflexive domains

- PRG plans to publish Technical Monographs
The Modular One and OS1...

- **Always** used with IC interpreter, **never** the raw machine

- OS
  
  "The most important single feature, however, is the hierarchical nature of its control structure, which avoids the need for a special job-control language."
  
  - "Job Control Language" should be (subset of) the programming language
  - Job invocation same as function call
    - Difference: in case of error abandon job

- Streams (and later files)
  
  "The input/output system uses a very general form of stream; the filing system is designed to have a clear and logical structure."
  
  - a stream characterized by operations available on it
  - file structure very similar to Unix's
    - inodes, hard and symbolic links, etc
    - (eventually) predecessor directories (form of version control)
A Stream

\[
\text{let } \text{Next}[S] = (S \downarrow \text{NEXT})[S]
\]

\[
\text{and } \text{Out}[S,x] \text{ be } (S \downarrow \text{OUT})[S,x]
\]

\[
\ldots
\]
The Modular One and OS1...

- **Always** used with IC interpreter, **never** the raw machine

- **OS**
  
  "The most important single feature, however, is the hierarchical nature of its control structure, which avoids the need for a special job-control language."
  
  - "Job Control Language" should be (subset of) the programming language
  
  - Job invocation same as function call
    
    - Difference: in case of error abandon job

- Streams (and later files)
  
  "The input/output system uses a very general form of stream; the filing system is designed to have a clear and logical structure."
  
  - a stream characterized by operations available on it
  
  - file structure very similar to Unix's
    
    - inodes, hard and symbolic links, etc
    
    - (eventually) predecessor directories (form of version control)
1969

- PRG gets its own computer
  - (32K store; paper tape; no disc)
  - Prepare system on KDF9 in Jan, Feb
  - One visit to Hemel Hempstead
  - Machine arrives in March; system working within 48hrs.

- Dana Scott visits in Michaelmas Term
  - Strachey meets Scott at WG2.2 in Vienna (April)
  - Long weekly seminars in Michaelmas Term
    1) “Type-theoretic alternative to CUCH, ISWIM, OWHY”
    2) More explicit representation and derepresentation
    3) Inverse limit construction for reflexive domains

- Plan PRG Technical Monographs
1969

FIP

SASEND

1970

TEAMS

JASON

1971

TEAMS

JASON

1969

System for福s.D. written in DP4. O

comp PCT (by Nuffield) into virtual machine code.

Model on arrives. System working with 48hour.

Rennan meeting Wg7.2. C5 meets D5.

Fermi & Knuth grant £160,000 to include computer

Gaia, 100,000 £100,820 for main language.

D. Scott at Oxford

Development of Theory of Reduced Domain

Start of Use of Simulated Reducible System. Test

Start Wm, 150co, DP5: 1, 4, 2, 3, 3

1970

Public Tech. Technical Report PRG 1

Pub. 1968

Work on Virtual Machines & Keen report STS 1

1971

Students: Nos: D: 5, DPA: 1, 4, 0 2, 4, 3 3, 2, 3

Pub: Inf. Tech. PRG 2, 3

Report on PRG by Illis Batei (0.988)


Students: Nos: D: 5, DPA: 1, 4, 0 2, 4, 3 3, 2, 3.
Later Work

• Theoretical
  − Continuations (starts 1972) (*Wadsworth*)
  − Semantics of Algol 60 (*Mosses*)
  − Monographs
  − Lecture courses
  − Proof Rules and Math. Semantics (*Ligler*)
  − Adams Essay (*Strachey and Milne*)

• Practical
  − OS6 / OSPub
  − Design of Instruction Sets (*McGregor*)
  − Semantics & Pragmatics of \( \lambda \)-calculus (graph-reduction) (*Wadsworth*)
  − Compiler Generator (*Mosses*)
  − (Implementation of PAL) (*Turner*)
    − →SASL→KRC→Miranda (→Haskell)
  − Computing Mechanisms (*Derret*)
PRG-10: The Varieties of Programming Language

PREFACE

(With apologies to Professor William James, Miss Stella Gibbons and the late Herr Baedeker.)

**In my belief that a large acquaintance with particulars often makes us wiser than the mere possession of abstract formulas, however deep, I have ended this paper with some concrete examples, and I have chosen these among the extreme designs of programming languages. To some readers I may consequently seem, by the time they reach the end of the paper, to offer a caricature of the subject. Such convulsions of linguistic purity, they will say, are not sane. It is my belief, however, that there is much of value to be learnt from the study of extreme examples, not least, perhaps, that our view of sanity is rather easily influenced by our environment; and this, in the case of programming languages, is only too often narrowly confined to a single machine. My ambition in this and other related papers, mostly so far unwritten, is to develop an understanding of the mathematical ideals of programming languages and to combine them with other principles of common sense which serve as correctives of exaggeration, allowing the individual reader to draw as moderate conclusions as he will.
(With apologies to Professor William James, Miss Stella Gibbons and the late Herr Baedeker.)

**In my belief that a large acquaintance with particulars often makes us wiser than the mere possession of abstract formulas, however deep, I have ended this paper with some concrete examples, and I have chosen these among the extreme designs of programming languages. To some readers I may consequently seem, by the time they reach the end of the paper, to offer a caricature of the subject. Such convulsions of linguistic purity, they will say, are not sane. It is my belief, however, that there is much of value to be learnt from the study of extreme examples, not least, perhaps, that our view of sanity is rather easily influenced by our environment; and this, in the case of programming languages, is only too often narrowly confined to a single machine. My ambition in this and other related papers, mostly so far unwritten, is to develop an understanding of the mathematical ideals of programming languages and to combine them with other principles of common sense which serve as correctives of exaggeration, allowing the individual reader to draw as moderate conclusions as he will.
Later Work

● Theoretical
  - Continuations (starts 1972) (*Wadsworth*)
  - Semantics of Algol 60 (*Mosses*)
  - Monographs
  - Lecture courses
  - Proof Rules and Math. Semantics (*Ligler*)
  - Adams Essay (*Strachey and Milne*)

● Practical
  - OS6 / OSPub
  - Design of Instruction Sets (*McGregor*)
  - Semantics & Pragmatics of $\lambda$-calculus (graph-reduction) (*Wadsworth*)
  - Compiler Generator (*Mosses*)
  - (Implementation of PAL) (*Turner*)
    → SASL → KRC → Miranda → Haskell
  - Computing Mechanisms (*Derret*)
Correspondence

To the Editor,
The Computer Journal.

An impossible program

Sir,
A well-known piece of folk-lore among programmers holds that it is impossible to write a program which can examine any other program and tell, in every case, if it will terminate or get into a closed loop when it is run. I have never actually seen a proof of this in print, and though Alan Turing once gave me a verbal proof (in a railway carriage on the way to a Conference at the NPL in 1953), I unfortunately and promptly forgot the details. This left me with an uneasy feeling that the proof must be long or complicated, but in fact it is so short and simple that it may be of interest to casual readers. The version below uses CPL, but not in any essential way.

Suppose $T[R]$ is a Boolean function taking a routine (or program) $R$ with no formal or free variables as its argument and that for all $R$, $T[R] = \text{True}$ if $R$ terminates if run and that $T[R] = \text{False}$ if $R$ does not terminate. Consider the routine $P$ defined as follows

rec routine $P$

$\text{§ } \text{L : if } T[P] \text{ go to } L$

Return $\text{§}$

If $T[P] = \text{True}$ the routine $P$ will loop, and it will only terminate if $T[P] = \text{False}$. In each case $T[P]$ has exactly the wrong value, and this contradiction shows that the function $T$ cannot exist.

Churchill College, Cambridge.

Yours faithfully,
C. Strachey.
Later Work

- **Theoretical**
  - Continuations (starts 1972) (*Wadsworth*)
  - Semantics of Algol 60 (*Mosses*)
  - Monographs
  - Lecture courses
  - Proof Rules and Math. Semantics (*Ligler*)
  - Adams Essay (*Strachey and Milne*)

- **Practical**
  - OS6 / OSPub
  - Design of Instruction Sets (*McGregor*)
  - Semantics & Pragmatics of $\lambda$-calculus (graph-reduction) (*Wadsworth*)
  - Compiler Generator (*Mosses*)
  - (Implementation of PAL) (*Turner*)
    - →SASL→KRC→Miranda →Haskell
  - Computing Mechanisms (*Derret*)
In fact, the invention of the computing machine is a technological advance of greater importance than any other since the invention of printing. In the long run the effect of having a mechanical assistant to our thinking will probably have as profound an influence on our intellectual life, and hence on our whole civilization, as the introduction of more or less universal literacy. Fortunately, however, I think it will be at least two hundred years before these changes become really significant—fortunately, I say, because like most other changes they are going to be very uncomfortable.
**What was he thinking? (1970)**

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Grades</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>State of Development</td>
</tr>
<tr>
<td></td>
<td>Relevance</td>
</tr>
<tr>
<td><strong>A. Machine Design</strong></td>
<td></td>
</tr>
<tr>
<td>1. Electronic Engineering, Hardware</td>
<td>(\gamma)</td>
</tr>
<tr>
<td>2. Machine Architecture, Logical Design</td>
<td>(\alpha \text{ or } \beta)</td>
</tr>
<tr>
<td><strong>B. Application</strong></td>
<td></td>
</tr>
<tr>
<td>3. Numerical Analysis</td>
<td>(\gamma)</td>
</tr>
<tr>
<td>4. Problem Solving</td>
<td>(\beta)</td>
</tr>
<tr>
<td>5. Artificial Intelligence</td>
<td>(\gamma)</td>
</tr>
<tr>
<td>6. Complicated Logical Structures</td>
<td>(\gamma)</td>
</tr>
<tr>
<td><strong>C. Computing</strong></td>
<td></td>
</tr>
<tr>
<td>7. Model Theory, e.g. Real Time and Simulation</td>
<td>(\alpha)</td>
</tr>
<tr>
<td>8. Programming Languages</td>
<td>(\alpha)</td>
</tr>
<tr>
<td>9. Computing Theory</td>
<td>(\alpha)</td>
</tr>
<tr>
<td>10. Syntax and Linguistics</td>
<td>(\gamma \text{ or } \beta \gamma)</td>
</tr>
<tr>
<td><strong>D. Mathematical Theory</strong></td>
<td></td>
</tr>
<tr>
<td>11. Mathematical Logic</td>
<td></td>
</tr>
<tr>
<td>(a) Decidability</td>
<td>(\beta)</td>
</tr>
<tr>
<td>(b) (\lambda)-calculus and Combinators</td>
<td>(\alpha \text{ or } \alpha \beta)</td>
</tr>
<tr>
<td>12. Information Theory</td>
<td>(\beta)</td>
</tr>
</tbody>
</table>

*Is Computing Science? — Girton Centenary Symposium, 1970*
What was he thinking? (1973)

1. Current State of Programming Language Theory
   1.1 Formal Semantics
   1.2 Proofs of Properties of Programs
   1.3 Programming Methodology

2. Influence of Programming Theory on Programming Practice
   2.1 Choice of Programming Language
   2.2 Size of Programs

3. Possible Technical Improvements
   3.1 Technical Problems and Myths
      (a) Unsuitable language facilities
      (b) Inefficient object code
      (c) Compiling slow
      (d) Inadequate debugging facilities
   3.2 Technical Remedies
      (a) Better languages
      (b) Better compilers and object code

4. Practical and Human Problems
   4.1 Programming Style
   4.2 Toughness

5. Prospects
   No overnight change
