, , 1–3 ()

Christopher Strachey: recollections of his influence

MIKE GORDON mjcg@cl.cam.ac.uk
University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street, Cambridge CB2 3QG, U.K.

Abstract. My early research was inspired by the mathematical semantics of Scott and Strachey. Two such topics, recounted in this paper, were the fixed-point analysis of pointer loops and the expressibility of a style of functional programming introduced by Barron and Strachey.

Keywords: Semantics, Programming.

I started work on my PhD at Edinburgh in the early 1970s, supervised by Rod Burstall. Shortly before, Dana Scott had shown how to put Strachey's approach to semantics on a sound mathematical footing. This exhilarating and elegant combination of programming language theory (Strachey) and sophisticated mathematics (Scott) generated tremendous excitement and was seen as a major breakthrough and a very hot topic for research.

In the late 1960s, Strachey and Burstall had shown how to model pointers with a mapping, called a store, from references (L-values) to values. The technique of 'tying a knot' with pointers was then a standard way to implement recursion and could be represented by constructing circular loops of pointers in a store. This technique raised the question of whether such loops could be characterised as least fixed-points.

The implementation of recursion with pointer loops is similar to the implementation of recursion in Lisp and POP-2 using dynamic binding. In these languages, the execution of procedure bodies takes place in the current environment in which the name of the procedure is bound to its body, so that executing a recursive call results in reentering the body in the current environment. Techniques developed by Strachey's student Chris Wadsworth [9] turned out to be suitable for showing the equivalence of a semantics of recursive Lisp procedures using knots to a semantics using fixed-points. A corollory was the equivalence of the Lisp EVAL algorithm to a Scott-Strachey style mathematical semantics [3].

Although my study of the relation between loops and fixed-points started in the first year of my PhD, I continued to pursue it as postdoctoral research at Stanford and made further progress [4] using the method of 'inclusive predicates' developed by another of Strachey's students, Robert Milne [6]. (Inclusive predicates were independently discovered by John Reynolds, who used them to relate direct and continuation semantics [7].) These early techniques of Milne and Reynolds developed into the method of logical relations, now a powerful tool for the analysis of both denotational and operational semantics.

Strachey's programming skills were legendary. Whilst at Cambridge, he devised a simple, yet extremely powerful, language for writing macros, called GPM (General Purpose Macrogenerator [8]). I wrote a mathematical semantics of this as my first

serious exercise in composing formal semantic descriptions. Alas, the details of this semantics have been lost.

Another product of Strachey's time at Cambridge was the 1966 paper entitled 'Programming', co-authored with David Barron [1]. This paper contains many surprisingly modern functional-programming techniques. It introduced the function *Lit*, now called `foldr` in ML and other modern functional languages. Strachey devised the following amazing definition of a function to compute the Cartesian product of a list of lists.

$$\begin{array}{l}\textbf{let } \mathit{Product}[L] \;=\; \mathit{Lit}[f, \mathit{List}1[\mathrm{NIL}], L] \\ \qquad \textbf{where } f[k,z] \;=\; \mathit{Lit}[g, \mathrm{NIL}, k] \\ \qquad\qquad \textbf{where } g[x,y] \;=\; \mathit{Lit}[h, y, z] \\ \qquad\qquad\qquad \textbf{where } h[p,q] \;=\; \mathit{Cons}[\mathit{Cons}[x,p],q]\end{array}$$

This program is written in the language CPL, which Strachey, Barron, Hartley and others were working on at the time. The definition of *Product* in ML would be

```
fun Product L
    = foldr (fn (k,z) =>
                foldr (fn (x,y) =>
                          foldr (fn (p,q) =>
                                    (x::p)::q) y z) [] k) [[]] L
```

For example, evaluating `Product [[1,2],[3,4],[5,6]]` yields `[[1,3,5],[1,3,6],[1,4,5],[1,4,6],[2,3,5],[2,3,6],[2,4,5],[2,4,6]]`

The question naturally arose of what could and couldn't be programmed just using *Lit*. Plotkin had shown (personal communication, Edinburgh, about 1972) that if the number n was represented by a list of n empty lists, then all primitive recursive functions could be programmed. My first ever paper (never published) investigated the limits of '*Lit*-computability' [2]. To this end, a little programming language was defined whose programs formalised Strachey-style combinations of *Lit*. A function was defined to be *Lit*-computable if it was denoted by a program in the language, via a Scott-Strachey semantics. Various results were obtained, for example that the Lisp list-equality function `equal` is not *Lit*-computable.

Strachey strongly influenced my early research and his ideas had a substantial impact on the course of my academic activities: the first class I ever taught was on denotational semantics and it gave rise to an early textbook describing the methods of Strachey and his students [5]. Twenty-five years later, I still find myself composing formal semantic specifications, e.g., recently for the Verilog hardware description language. Without Strachey's influence my professional life would have been very different. I met him twice. On one of the occasions he listened sympathetically to my half-baked ideas about relating knots and fixed points even though I was only a very junior research student. Although I only knew him slightly, I owe him a lot.

Acknowledgments

Olivier Danvy and Andrzej Filinski made numerous suggestions to help develop this note from a very rough first draft.

References

1. D.W. Barron and C. Strachey. Programming. In L. Fox, editor, *Advances in Programming and Non-numerical Computation*, pages 49–82. Pergammon Press, 1966.
2. M. J. C. Gordon. An investigation of *Lit*. Technical Report Memorandun MIP-R-101, University of Edinburgh, School of Artificial Intelligence, May 1973.
3. M. J. C. Gordon. Operational reasoning and denotational semantics. Technical Report Computer Science Department, Report No. STAN-CS-75-506, Stanford University, August 1975.
4. M. J. C. Gordon. Towards a semantic theory of dynamic binding. Technical Report Computer Science Department, Report No. STAN-CS-75-507, Stanford University, August 1975.
5. M. J. C. Gordon. *The Denotational Description of Programming Languages*. Springer-Verlag, 1979.
6. R. E. Milne. *The Formal Semantics of Computer Languages and Their Implementations*. PhD thesis, The University of Cambridge, 1974. (Note: Milne obtained his PhD from Cambridge, but he worked under Strachey at Oxford).
7. J. C. Reynolds. On the relation between direct and continuation semantics. In J. Loeckx, editor, *Proceedings of the Second Colloquium on Automata, Languages and Programming*, pages 141–156. Springer-Verlag, 1974.
8. C. Strachey. A general purpose macrogenerator. *The Computer Journal*, 8(6):225–241, 1965.
9. C. P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞ models of the lambda-calculus. *SIAM Journal of Computing*, 5:488–521, 1976.