Higher-Order and Symbolic Computation, 13, 51–55, 2000
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

Christopher Strachey—Understanding Programming Languages

ROD BURSTALL rb@dcs.ed.ac.uk
Informatics, Edinburgh University, UK

It is hard to start writing this. Perhaps I am still somewhat in awe of Christopher, even though it is almost twenty-five years since he died.

I got to know him through Peter Landin. I had met Landin via a chance meeting in Foyle's Bookshop with Mervyn Pragnell, a charmingly eccentric character who ran an underground study group in logic at Birkbeck College, in London—I say underground because the College knew nothing of our nocturnal visits, only made possible by a Birkbeck Lab Assistant who had a key. Landin taught me about lambda calculus and functional programming in the pub round the corner from the College, The Duke of Marlborough. He was working as a consultant, an assistant to a superior sort of chap called Christopher Strachey. When I quit my job as a programmer in a paper company to do a Ph.D. at Birmingham, I continued to see Landin and on occasion this involved a sortie to Kensington where he worked in a back room of Strachey's imposing house. I made my way from the front door, rather hoping not to bump into Strachey in the corridor.

Towards the end of my two years in Birmingham, I received a phone call from a Donald Michie of Edinburgh University who asked if I might be interested in a job in his embryonic "Experimental Programming Unit". As he proposed an interview in a Chinese restaurant in Birmingham, I was reluctant to turn him down. Thus lured to Edinburgh for further discussion of the proposal, I found myself conducted to the top of a hill, Arthur's Seat, with a beautiful view of the city; I accepted the job halfway down without actually having seen the University. Donald subsequently broke the news that he had asked Strachey to tip him off about likely candidates, promising that in return the new employee would be indentured to labour for Strachey for the first three months in Kensington.

On this basis I returned to the Strachey house in October 1965 to be entertained, now in the front room, by Strachey himself with a glass of Madeira and talk of a wondrous "Combined Programming Language".

Strachey told me about two projects. The first was the definition and implementation of CPL with help from Cambridge and London Universities; he also wanted to create a simpler version of CPL which would be easier to implement.

The second project was the semantics of programming languages. He was trying to give semantics to imperative languages using combinators: as I remember he wanted to combine operations which both produced a result and changed the state (perhaps this was a distant precursor of Moggi's use of monads with their Kleisli composition to handle expressions with side effects).

CPL and its posterity

I did some work on CPL, and in 1965 I wrote a report at Edinburgh, now long vanished, on "Some Aspects of CPL Semantics". There were two implementations of CPL, but it was indeed rather too ambitious for the limited machine resources of the time, especially when the Atlas/Titan computer line was discontinued after a few machines had been built. However, many of us learnt from the CPL concepts, amongst others left-hand values (locations—an advance on Algol 60's use of call-by-name) and procedures used as both parameters and results. (Strachey called this the principle of "functions as first-class citizens"). Above all I learned about strong typing, particularly for record types and about polymorphism.

At that time I had been using Algol 60 for my thesis work on heuristic programs. LISP I knew about through Landin, and of course Landin's own functional language ISWIM (If You See What I Mean) since its Duke of Marlborough prototype version. Landin had already written about the SECD machine [8] and given a translation from Algol 60 to lambda calculus plus assignment [9]. I continued to learn a great deal from him and collaborate with him. Landin taught me about functional languages and formal language definition and Strachey extended these to a workable imperative language, with an analysis of the role of assignment and locations. Strachey wanted implementations whereas Landin worked on paper (or beer mats).

Strachey's idea of a simplified CPL came to full flower in Martin Richards's BCPL language at Cambridge which was eventually transmuted into C at Bell Labs. Thus CPL was a legitimate ancestor of the world's most influential programming language, used in implementing many other languages and operating systems.

CPL had another less eminent descendant, also influenced by LISP and ISWIM, namely the POP-2 language developed at Edinburgh University by Robin Popplestone and myself [5, 6]. Robin came to Edinburgh in 1965 with a LISP influenced language which we renamed POP-1 (he called it COWSEL which sounded to us too rural for a programming language). He and I redesigned it as POP-2 with a more Algol style syntax and a number of Landin/Strachey derived features:

- closures, enabling functions to be first-class citizens (constructed explicitly in POP-2 by 'partial application').
- a generalisation of Strachey's notion of load-update pair, called doublet. Instead of a location and a value a doublet is a pair of functions, a set function and a get function. The set function takes *n* arguments and produces no result, and the get function takes *n* arguments and produces a result. When the doublet is applied to *n* arguments, it calls the set if it is on the left of an assignment and calls the get if it is on the right. So you can regard an array, for example, as a pair of functions, where the arguments are the array indices; you could make an association list look like an array.
- records with run-time type checking (including lists). Compile-time type checking was beyond our resources at that time, especially in an interactive system.
- a jumpout device for handling exceptions.
- dynamic lists, modelled on Landin's streams.

POP-2 was used for research and teaching in several British universities, notably Edinburgh and Sussex, over a long period. It never spread to the U.S. where LISP reigned supreme in AI circles. However it was used for three notable achievements in Edinburgh:

- the second time sharing system to be created in the UK (Cambridge produced the first) [11] (Strachey was credited by Fano and Corbatò of MIT Project MAC with suggesting the idea of time sharing in 1958 [7], although it seems that he was concerned with a single-user form of sharing between a main process and peripherals rather than the multi-user systems pioneered by Project MAC.)
- the Boyer-Moore theorem prover [2]
- the first real robot assembly system—it recognised parts and put together wooden toys, a boat and a car (the previous best was building a pile of bricks) [1]

Strachey also influenced the implementation of POP-2. He impressed on me the need for a sophisticated "link loader", to enable one to connect independently compiled pieces of code. This was before the days of "make" in C. His programming abilities were demonstrated when he and colleagues in Oxford wrote an operating system for the Modular one machine and got it working within a week or so of delivery.

Semantics

Strachey's challenge to give semantics to realistic programming languages stayed with me for a long time. Landin had written a semantics of Algol 60 as an informal but precise translation into lambda calculus extended with assignment. My first effort was to improve this by translating to lambda calculus with let instead of assignment. I wrote a paper called "The Semantics of Assignment" [3] for the first of a series of Machine Intelligence Workshops organised at Edinburgh by Donald Michie, and attended by such luminaries as John McCarthy and Alan Robinson.

More significant was a semantics for a large part of Algol 60 which I gave by an informal translation to first order logic [4]. It axiomatised a "next state" function. It represented the program by a relation between statements (terms) rather than as a single term as in Plotkin's later structured operational semantics [10] because I was still accounting for goto's and labels which later became "deprecated" ("considered harmful"). Nonetheless it was a formal operational semantics and covered many features of Algol, such as arrays, recursive procedures and side-effects; it also covered list processing with assignments to head and tail. I implemented a simple proof checker and succeeded in doing a correctness proof for a simple program using my semantics. Thus I believed that I had fulfilled Strachey's request for a formal semantics for a realistic language.

However at almost the same time, Scott came up with domain semantics for the untyped lambda calculus. Strachey enthusiastically espoused this as a means of giving denotational semantics to real programming languages. This work was very influential in Edinburgh through the efforts of Gordon Plotkin, Michael Gordon (my second Ph.D. student) and Robin Milner. Michael Gordon's thesis was on a denotational semantics for LISP, mainly guided by Plotkin. This all led to the LCF proof system based on Scott domains and LCF's influential metalanguage ML. Michael Gordon took LCF to Cambridge and transformed it into the HOL proof checker.

Abstraction

One of Strachey's ideas which I found most inspiring was his use of abstraction in describing programming language and data. His work on semantics concerned abstraction for programming languages, but he also firmly advocated abstraction in describing data. In 1966 he wrote a very lucid article for *Scientific American* [12] in which he gave a development of a program in informal CPL for playing checkers. He started with an abstract notion of board state and then refined it to two different representations, an obvious one and an efficient one. It is a beautiful exposition of how to write a program, still valid today.

This influenced my own thinking strongly. Eventually Strachey's intuitions emerged in the community as the notion of abstract data type, with the distinction which he had made between an abstract description of data and its representation.

Envoi

Strachey continued to be a good friend and mentor to me as a young researcher in the sixties and early seventies. His elegance of manner was accompanied by an elegance of thought and language which was a continual inspiration. In late 1974 he visited Edinburgh to give a talk to a densely packed seminar room. I knew that he was working very hard on his book on semantics, with his co-author Robert Milne. I was shocked to hear not long after that he had died suddenly. One memory endures:

In 1967 at the Programmming Research Group, David Park picked up the phone. A voice said "This is Buckingham Palace". David laughed, but it was indeed Buckingham Palace, inviting Christopher to have lunch with the Queen. A few months later, at the Copenhagen Summer School, Christopher delighted our three small daughters by telling them about the lunch. They were particularly interested in the fact that the Queen had taken her shoes off under the table. The next day they presented him with a drawing of the occasion. There was a very long table with the Queen sitting at one end wearing a crown, but no shoes, and Christopher sitting at the other end, with shoes and also wearing a crown.

Why not? I think he deserved a moment of glory.

References

1. Ambler, A.P., Barrow, H.G., Brown, C.M., Burstall, R.M., and Popplestone, R.J. A versatile system for computer-controlled assembly. *Artificial Intelligence* **6** (1975) 129–156.
2. Boyer, R.S. and Moore, J.S. Proving theorems about LISP functions. *Journal of the ACM* **22**(10) (1975) 129–144.
3. Burstall, R.M. Semantics of assignment. In *Machine Intelligence*, Vol. 2, E. Dale and D. Michie (Eds.). Edinburgh University Press, 1968, pp. 3–20.
4. Burstall, R.M. Formal description of program structure and semantics in first order logic. In *Machine Intelligence*, Vol. 5, B. Meltzer and D. Michie (Eds.). Edinburgh University Press, 1969, pp. 79–98.
5. Burstall, R.M., Collins, J., and Popplestone, R.J. *Programming in POP-2*. Edinburgh University Press, 1971.

6. Burstall, R.M., and Popplestone, R.J. POP-2 reference manual. In *Machine Intelligence*, Vol. 5, B. Meltzer and D. Michie (Eds.). Edinburgh University Press, 1968, pp. 207–246.
7. Fano, R.M. and Corbatò, F.J. Time-sharing on computers. *Scientific American* **215**(3) (1966) 129–140.
8. Landin, P.J. The mechanical evaluation of expressions. *Computer Journal* **6** (1964) 308–320.
9. Landin, P.J. A correspondence between ALGOL 60 and Church's lambda notation. *Communications of the ACM* **8** (1965) 89–101 and 158–165.
10. Plotkin, G.D. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, September 1981.
11. Pullin, D. A plain man's guide to Multi-POP implementation. Mini-MAC Report 2, Department of Machine Intelligence and Perception, Edinburgh University, 1967.
12. Strachey, C. Systems analysis and programming. *Scientific American* **25**(3) (1966) 112–124.