Peter J. Landin (1930–2009)

Olivier Danvy
Department of Computer Science,
Aarhus University
Aabogade 34, DK-8200 Aarhus N, Denmark
danvy@cs.au.dk

Abstract

This note is a prelude to a forthcoming special issue of HOSC dedicated to Peter Landin's memory.

—

One of the founding fathers of everything lambda in programming languages passed away in June 2009: Peter J. Landin.

Peter Landin spent the last years of his life as Professor Emeritus at Queen Mary, where his colleagues included Edmund Robinson and Peter O'Hearn. Over the last decade, he served in the advisory board of HOSC and thus received a complimentary copy of each issue. HOSC published two of his articles: a tribute to Christopher Strachey [21] in 2000 and a reprint of his 1965 technical note "A generalization of jumps and labels" [20] in 1998, for which Hayo Thielecke wrote an introduction [30]. In the editorial of our 1998 issue [11], Carolyn Talcott and I presented this reprint as follows:

> This paper describes a real conceptual discovery, namely the idea to make control facilities first-class entities in a programming language, through the "J operator." Its exposition is typical of the simplicity, directness, clarity and honesty of Landin's writing that makes his articles such a pleasure to read.

This spring, for the last time, I sent him a copy of a scientific compliment: a joint re-visitation with Ken Shan and Ian Zerny of his direct-style embedding of Algol 60 into applicative expressions with the J operator [15]. There, we retarget his embedding to the Rhino implementation of JavaScript with continuation objects. Indeed, whereas call/cc captures the current continuation, the J operator captures the continuation of the caller of the current method [7]. This feature fitted Landin's embedding then and it fits a JavaScript implementation with a local stack for each method now [3]. Playfully, the title of our re-visitation is thus "J is for JavaScript" [10].

—

In 2004, I paid Peter Landin a visit at the occasion of Josh Berdine's PhD defense [2] and found him in his office, patiently helping an undergraduate student. In turn, I patiently waited for him to be done with the student before presenting him my rational deconstruction of his SECD machine [5]. I then showed him how the SECD machine could be put into defunctionalized form [9, 25] and could then be refunctionalized [8] into a continuation-passing evaluator à la Lockwood Morris [22]. I thus enthusiastically concluded how much the SECD machine made sense, and that even though he might not have discovered continuation-passing style (see Appendix), defunctionalization and refunctionalization provided a concrete argument why his name should be added to the list of the discoverers of continuations [26]. Throughout, he was as patient with me as with the undergraduate student, and in the end he smiled, his eyes sparkled amusedly, and then he made some incredibly modest comments to the effect that he had been lucky.

Peter Landin was indeed so modest that in 1998, he did not attend the MFPS XIV session held in his honor at Queen Mary,[1] eliciting Dana Scott's quip as to whether Peter Landin was the Bourbaki of Computer Science. He did, however, get to read hardcopies of the slides displayed at his session.

—

I initially got in touch with Peter Landin in 1996 by e-mail and by phone and we met for the first time in January 1997 in Paris, at the occasion of the Second ACM SIGPLAN Workshop on Continuations [4], which I was chairing and where he gave a keynote speech. For the proceedings, he wrote the masterfully idiosyncratic "Histories of

[1] <http://www.dcs.qmul.ac.uk/~edmundr/mfps/>

Discoveries of Continuations: Belles-Lettres with Equivocal Tenses" [19].[2]

After his keynote speech at CW'97, he handed out copies of some of his old research reports [16, 17]. There naturally was a stampede, and to Olin Shivers who asked his copies to be autographed he said "I am not the Beatles," and then signed them.

When introducing him before his keynote speech, I pointed out that independently of all his accomplishments, he was a rare breed of computer scientist with a control operator as his middle name (which is "John" and is abbreviated "J" in his publications). He flashed a look at me that to this day makes me wonder whether it was such a good idea to mention this coincidence at all.

My favorite moment with Peter Landin occurred when we met: he was arriving from London to attend CW'97, I picked him up at the train station, and together with John Reynolds and Andrzej Filinski, we sat at the terrace of a French café. I took the opportunity of a pause in the conversation to venture the question as to whether in their mind, the evaluation order of the meta-language of denotational semantics was call by value or call by name. Peter and John immediately, and simultaneously, answered "call by value of course" (for Peter) and "call by name of course" (for John). For a second of eternity, they looked at each other. Then it was like they were mentally telling each other "let's not have this discussion again" and the universe resumed its course. The rest of the evening was warm and pleasant, the following day was as wonderful as each continuation workshop somehow manages to be, and eventually I took him back to the train station.

—

What happened before is history: his impression that computer science was turning "too theoretical" for him, his quiet move away from the programming-language limelight, and his ascension to programming-language legend. Peter Landin was indeed gifted with an uncanny, almost prophetic, computational sense. To (boldly) quote from the introduction of my rational deconstruction of his SECD machine [5]:

> Forty years ago, Peter Landin wrote a profoundly influential article, "The Mechanical Evaluation of Expressions" [14], where, in retrospect, he outlined a substantial part of the functional-programming research programme for the following decades. This visionary article stands out for advocating the use of the λ-calculus as a meta-language and for introducing the first abstract machine for the λ-calculus (i.e., in Landin's terms, applicative expressions), the SECD machine. However, and in addition, it also introduces the notions of 'syntactic sugar' over a core programming language; of 'closure' to represent functional values; of circularity to implement recursion; of thunks to delay computations; of delayed evaluation; of partial evaluation; of disentangling nested applications into where-expressions at preprocessing time; of what has since been called de Bruijn indices; of sharing; of what has since been called graph reduction; of call by need; of what has since been called strictness analysis; and of domain-specific languages—all concepts that are ubiquitous in programming languages today.

[2] Including "So these continuations have continuations." which beautifully anticipates the CPS hierarchy [6].

And did I mention that together with his embedding of Algol 60 into applicative expressions, his 700 article [18] is generally recognized as the origin of domain-specific languages today?

On so many fundamental and tasteful ways Peter Landin was unerringly right. He has now passed away, but his writings stay and his discoveries, his inventions, and his middle name live on.

—

Appendix: As John Reynolds pointed out in our columns [26], Peter Landin did not discover continuation-passing style—instead, he invented control operators and first-class continuations (see Figure).

The left vertical arrow is a tour de force due to Adriaan van Wijngaarden [31] and James Morris [23]. The top horizontal arrow is due to Peter Landin [15]. ("Applicative expressions" is Peter Landin's words for "λ-terms.") The diagonal arrow is variously due to Christopher Strachey and Christopher P. Wadsworth [29] and to Kamal Abdali [1], and its unstaged version is due to Lockwood Morris in the form of a definitional interpreter in continuation-passing style [22, 25]. In the pure case (i.e., without the J operator), the right vertical arrow is due to Michael Fischer [12] and has been formalized by Gordon Plotkin [24] and put to compiler use by Guy Steele [28], who extended it to the impure case and introduced the acronym "CPS" and the term "CPS transformation." The bottom horizontal arrow is obvious. In Landin's direct-style embedding, label declarations are mapped to an occurrence of the J operator that gives rise to a 'program closure' (known today as a "first-class continuation" [13]), and jumps to a label are mapped to an application of the program closure lexically associated to this label. Peter Landin used to joke that he had smuggled the J operator into a galley proof [15].

"In those days [the 1960's],
many successful projects started out
as graffitis on a beer mat
in a very, very smoky pub."
Peter J. Landin, 2004

Acknowledgments: This note benefited from Irène Danvy, Julia Lawall, Karoline Malmkjær, and Ian Zerny's sensible proof-reading.

$$\begin{tikzcd}
\begin{array}{c}\text{Algol programs}\\\text{in direct style}\end{array} \arrow[r, "\text{direct-style}", "\text{embedding}"'] \arrow[d, dashed, "\begin{array}{c}\text{CPS}\\\text{transformation}\end{array}"'] \arrow[dr, "\begin{array}{c}\text{CPS}\\\text{embedding}\end{array}" description] & \begin{array}{c}\text{applicative expressions}\\\text{in direct style with J}\end{array} \arrow[d, dashed, "\begin{array}{c}\text{CPS}\\\text{transformation}\end{array}"] \\
\begin{array}{c}\text{Algol programs}\\\text{in CPS}\end{array} \arrow[r, dotted, "\text{direct-style}", "\text{embedding}"'] & \begin{array}{c}\text{applicative expressions}\\\text{in CPS}\end{array}
\end{tikzcd}$$

References

[1] S. Kamal Abdali. A lambda-calculus model of programming languages, part II: Jumps and procedures. *Computer Languages*, 1(4):303–320, 1976.

[2] Josh Berdine. *Linear and Affine Typing of Continuation-Passing Style*. PhD thesis, Queen Mary, University of London, 2004.

[3] John Clements, Ayswarya Sundaram, and David Herman. Implementing continuation marks in JavaScript. In Will Clinger, editor, *Proceedings of the 2008 ACM SIGPLAN Workshop on Scheme and Functional Programming*, pages 1–9, Victoria, British Columbia, September 2008.

[4] Olivier Danvy, editor. *Proceedings of the Second ACM SIGPLAN Workshop on Continuations (CW'97)*, Technical report BRICS NS-96-13, Aarhus University, Paris, France, January 1997.

[5] Olivier Danvy. A rational deconstruction of Landin's SECD machine. In Clemens Grelck, Frank Huch, Greg J. Michaelson, and Phil Trinder, editors, *Implementation and Application of Functional Languages, 16th International Workshop, IFL'04*, number 3474 in Lecture Notes in Computer Science, pages 52–71, Lübeck, Germany, September 2004. Springer-Verlag. Recipient of the 2004 Peter Landin prize. Extended version available as the research report BRICS RS-03-33.

[6] Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

[7] Olivier Danvy and Kevin Millikin. A rational deconstruction of Landin's SECD machine with the J operator. *Logical Methods in Computer Science*, 4(4:12):1–67, November 2008.

[8] Olivier Danvy and Kevin Millikin. Refunctionalization at work. *Science of Computer Programming*, 74(8):534–549, 2009. Extended version available as the research report BRICS RS-08-04.

[9] Olivier Danvy and Lasse R. Nielsen. Defunctionalization at work. In Harald Søndergaard, editor, *Proceedings of the Third International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'01)*, pages 162–174, Firenze, Italy, September 2001. ACM Press. Extended version available as the research report BRICS RS-01-23.

[10] Olivier Danvy, Chung-chieh Shan, and Ian Zerny. J is for Javascript: A direct-style correspondence between Algol-like languages and Javascript using first-class continuations. In Walid Taha, editor, *Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009*, number 5658 in Lecture Notes in Computer Science, pages 1–19, Oxford, UK, July 2009. IFIP, Springer.

[11] Olivier Danvy and Carolyn L. Talcott, editors. *Special Issue on the Second ACM Workshop on Continuations (CW 1997), Part I*, volume 11, number 2 of *Higher-Order and Symbolic Computation*, 1998.

[12] Michael J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, 1993. Available at `<http://www.brics.dk/~hosc/vol06/03-fischer.html>`. A preliminary version was presented at the ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

[13] Daniel P. Friedman and Christopher T. Haynes. Constraining control. In Mary S. Van Deusen and Zvi Galil, editors, *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 245–254, New Orleans, Louisiana, January 1985. ACM Press.

[14] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[15] Peter J. Landin. A correspondence between Algol 60 and Church's lambda notation, Parts 1 and 2. *Communications of the ACM*, 8:89–101 and 158–165, 1965.

[16] Peter J. Landin. A generalization of jumps and labels. Research report, UNIVAC Systems Programming Research, 1965. Reprinted in Higher-Order and Symbolic Computation 11(2):125–143, 1998, with a foreword [30].

[17] Peter J. Landin. Getting rid of labels. Research report, UNIVAC Systems Programming, July 1965.

[18] Peter J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, 1966.

[19] Peter J. Landin. Histories of discoveries of continuations: Belles-lettres with equivocal tenses. In Danvy [4], pages 1:1–9.

[20] Peter J. Landin. A generalization of jumps and labels. *Higher-Order and Symbolic Computation*, 11(2):125–143, 1998. Reprinted from a technical report, UNIVAC Systems Programming Research (1965), with a foreword [30].

[21] Peter J. Landin. My years with Strachey. *Higher-Order and Symbolic Computation*, 13(1/2):75–76, 2000.

[22] F. Lockwood Morris. The next 700 formal language descriptions. *Lisp and Symbolic Computation*, 6(3/4):249–258, 1993. Reprinted from a manuscript dated 1970.

[23] James H. Morris Jr. A bonus from van Wijngaarden's device. *Communications of the ACM*, 15(8):773, August 1972.

[24] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[25] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of 25th ACM National Conference*, pages 717–740, Boston, Massachusetts, 1972. Reprinted in Higher-Order and Symbolic Computation 11(4):363-397, 1998, with a foreword [27].

[26] John C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–247, 1993.

[27] John C. Reynolds. Definitional interpreters revisited. *Higher-Order and Symbolic Computation*, 11(4):355–361, 1998.

[28] Guy L. Steele Jr. Rabbit: A compiler for Scheme. Master's thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978. Technical report AI-TR-474.

[29] Christopher Strachey and Christopher P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. Technical Monograph PRG-11, Oxford University Computing Laboratory, Programming Research Group, Oxford, England, 1974. Reprinted in Higher-Order and Symbolic Computation 13(1/2):135–152, 2000, with a foreword [32].

[30] Hayo Thielecke. An introduction to Landin's "A generalization of jumps and labels". *Higher-Order and Symbolic Computation*, 11(2):117–124, 1998.

[31] Adriaan van Wijngaarden. Recursive definition of syntax and semantics. In T. B. Steel, Jr., editor, *Formal Language Description Languages for Computer Programming*, pages 13–24. North-Holland, 1966.

[32] Christopher P. Wadsworth. Continuations revisited. *Higher-Order and Symbolic Computation*, 13(1/2):131–133, 2000.