LISP AND SYMBOLIC COMPUTATION: An International Journal, 6, 233–247, 1993

The Discoveries of Continuations

JOHN C. REYNOLDS (*John.Reynolds@cs.cmu.edu*)
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Keywords: Semantics, Continuation, Continuation-Passing Style

Abstract. We give a brief account of the discoveries of continuations and related concepts by A. van Wijngaarden, A. W. Mazurkiewicz, F. L. Morris, C. P. Wadsworth, J. H. Morris, M. J. Fischer, and S. K. Abdali.

In the early history of continuations, basic concepts were independently discovered an extraordinary number of times. This was due less to poor communication among computer scientists than to the rich variety of settings in which continuations were found useful: They underlie a method of program transformation (into continuation-passing style), a style of definitional interpreter (defining one language by an interpreter written in another language), and a style of denotational semantics (in the sense of Scott and Strachey). In each of these settings, by representing "the meaning of the rest of the program" as a function or procedure, continuations provide an elegant description of a variety of language constructs, including call by value and **goto** statements.

1. The Background

In the early 1960's, the appearance of Algol 60 [32, 33] inspired a ferment of research on the implementation and formal definition of programming languages. Several aspects of this research were critical precursors of the discovery of continuations.

The ability in Algol 60 to jump out of blocks, or even procedure bodies, forced implementors to realize that the representation of a label must include a reference to an environment. According to Peter Naur:

> ... in order to specify a transfer of control we must in general supply both the static description of the destination ... and a dynamic description of its environment, the stack reference. This set ... together define what we call a program point. [31]

In retrospect, a program point was the representation of a continuation.

A more subtle realization was that return addresses could be treated on the same footing as procedure parameters. With a prescient choice of words, E. W. Dijkstra remarked:

> We use the name "parameters" for all the information that is presented to the subroutine when it is called in by the main program; function arguments, if any, are therefore parameters. The data grouped under the term "link" are also considered as parameters; the link comprises all the data necessary for the continuation of the main program when the subroutine has been completed. [9]

Indeed, in the GIER Algol Compiler designed by Naur and Jørn Jensen [31], return addresses and label parameters, both regarded as program points, were treated in essentially the same way.

Another precursor of continuations occurred in Peter Landin's SECD machine [14], a state-transition interpreter for a language of applicative expressions that was syntactically similar to the untyped lambda calculus but used a call-by-value order of evaluation. As captured by the acronym SECD, the state of the interpreter consisted of four components: a *stack*, an *environment*, a *control*, and a *dump*. The dump encoded the remaining computation to be executed after the control was exhausted; in retrospect it was another representation of a continuation.

To be able to translate Algol 60 into applicative expressions, Landin later extended these expressions and their interpreter with an assignment operation, and also a control operator J used to express the translation of **goto**'s and labels [15, 16]. In the extended SECD machine, the result of applying J was a value containing a dump. Thus, in modern terminology, the J operator provided a means of embedding continuations in values — and was an ancestor of operations such as Reynolds's **escape** [36], and `catch` [44] and `call/cc` [8] in Scheme.

2. A. van Wijngaarden

Apparently, the earliest description of a use of continuations was given by Adriaan van Wijngaarden (Director of the Mathematisch Centrum in Amsterdam) in September 1964, at an IFIP Working Conference on Formal Language Description Languages held in Baden bei Wien, Austria. A written version of this talk, along with a transcript of the discussion that followed, appears in the conference proceedings [45].

Van Wijngaarden's goal was to formulate a preprocessor that would translate Algol 60 into a more restricted sublanguage. The final stage of the preprocessing was (what we would now call) a transformation of proper procedures into continuation-passing style (CPS) [41], with an attendant elimination of labels and **goto** statements. (An earlier stage of the preprocessing replaced function procedures by proper procedures.)

As van Wijngaarden described the transformation:

> Provide each procedure declaration with an extra formal parameter — specified **label** — and insert at the end of its body a **goto** statement leading to that formal parameter. Correspondingly, label the statement following a procedure statement, if not labeled already, and provide that label as the corresponding extra actual parameter. [45]

Next, by inserting labels and **goto** statements, each block was transformed into an equivalent block with the form

$$\mathbf{begin}\ L_1\colon S_1;\ \ldots\ ; L_n\colon S_n\ \mathbf{end}$$

where every path of execution through a statement S_i ends in a **goto** statement. Actually, unless I have misunderstood his rather opaque prose, van Wijngaarden's description of this transformation is erroneous: When the above block is a statement of a larger block, there is no provision for inserting jumps from within the inner block to the statement following the inner block. This defect is easily remedied, however, by replacing such inner blocks by calls of corresponding parameterless procedures (as in the work of J. H. Morris [28]) before applying van Wijngaarden's transformation.

Finally, according to van Wijngaarden [45]:

> It is now completely harmless to insert at the end of each block an unlabeled **goto** statement leading to the first statement of that block, since this statement will never be executed. So far, we have only increased the number of labels and **goto** statements. But now we can perform the following operations:
>
> 1. Write before each label **procedure**.
> 2. Replace the colon following it by a semicolon.
> 3. Strike each **goto**.

The following exchange during the ensuing discussion shows that van Wijngaarden understood clearly that the transformation into continuation-passing style is more than just a method for eliminating labels and **goto**'s:

> MCILROY: ... I'm afraid you went a bit too far in the elimination of the **goto**, because this actually changes the temporal existence of values. If every **goto** is replaced by a procedure call, then this means that the entire history of the computation must be maintained. I'm a bit concerned about this limitation.
>
> VAN WIJNGAARDEN: I suppose you have a certain implementation of a procedure call in mind when you say that. But this implementation is only so difficult because you have to take care of the **goto** statement. However, if you do this trick I devised, then you will find that the actual execution of the program is equivalent to a set of statements; no procedure ever returns because it always calls for another one before it ends, and all of the ends of all of the procedures will be at the end of the program: one million or two million ends. If one procedure gets to the end, that is the end of all; therefore, you can stop. That means you can make the procedure implementation so that it does not bother to enable the procedure to return. That is the whole difficulty with procedure implementation. That's why this is so simple; it's exactly the same as a **goto**, only called in other words. [45]

In retrospect, it may seem surprising that van Wijngaarden's presentation did not make continuations and continuation-passing style into standard concepts of computer science. Participants in the discussion of his presentation included Dijkstra, Hoare, McCarthy, McIlroy, and Strachey, and other conference attendees included Böhm, Elgot, Landin and Nivat. But the idea didn't take hold. In particular, although Landin referred to van Wijngaarden's transformation in his own treatment of Algol 60 [15], he made no mention of the work when he heard F. L. Morris's colloquium in 1970. (See Section 4.) Moreover, Christopher Strachey never connected the work with Wadsworth's continuations, and did not cite van Wijngaarden in his own descriptions of the latter [42, 43]. (See Section 5.)

The discussion following the presentation reveals deep philosophical differences between van Wijngaarden and other researchers, particularly van Wijngaarden's abhorrence of abstract syntax and his belief that proper procedures were more basic than functions. A stronger barrier to communication was probably his failure to motivate the CPS transformation. According to M. D. McIlroy:

> I remember the talk well as an insightful tour de force of reductionism. ... Van Wijngaarden's argument shone clearly and unforgettably. ... [But] an idea can be understood without all

> its ramifications being seen, even by its originator. Since van Wijngaarden offered no practical examples of continuation passing, nor any theoretical application, save as a trick for proving one isolated and already known result, the value of continuations per se did not come through. [22]

Moreover, van Wijngaarden's discovery was actually the CPS transformation rather than continuations themselves — nowhere does he define or otherwise emphasize the actual concept of a continuation. This surely made it difficult to recognize any connection with the appearance of continuations in other settings.

It also appears that van Wijngaarden himself never used continuations again.

A final sidelight on the talk is remembered by McIlroy [22]:

> The talk actually had one direct and important consequence for computing. Under the inspiration of the notion of the unnecessity of **goto**'s, Dijkstra spent that evening constructing realistic examples of programs without **goto**'s, which he scribbled on napkins at coffee break the next day. In that exercise, he posited a "quit" statement (the `break` of CPL and C) .... That coffee-break palaver ripened into the most celebrated letter to the editor in the history of computing [10]. So while van Wijngaarden said that **goto**'s were unnecessary ..., Dijkstra stretched the point to say that **goto**'s were inconvenient. The latter lesson stuck.

3. A. W. Mazurkiewicz

In December 1969, Antoni W. Mazurkiewicz (then at the Instytut Maszyn Matematycznych in Warsaw) circulated a working paper entitled "Proving Algorithms by Tail Functions" to the membership of IFIP Working Group 2.2 (which included Strachey). A publication version was submitted to Information and Control in April 1970, revised in November, and published in April 1971 [18].

In this paper, Mazurkiewicz dealt with an automaton-like concept of an algorithm, consisting of a set E of labels, a subset $T \subseteq E$ of terminal labels, a set R of states, and a partial transition function γ from $(E - T) \times R$ to $E \times R$. He proposed that the semantics of such an algorithm, which he called a *tail function*, should be the least partial function φ from $E \times R$ to R satisfying

$$\varphi(e, x) = \begin{cases} \varphi(\gamma(e, x)) & \text{if } e \notin T \\ x & \text{if } e \in T\,. \end{cases}$$

When curried, the tail function is an environment mapping labels into command continuations, just as in the continuation semantics of an imperative language with labels and **goto** commands. Mazurkiewicz's concept of algorithm, however, is such a limited programming language — one without any hierarchical structure — that his work does not reveal much of the general nature of continuations. For example, there is no syntactic entity whose meaning must be a function accepting continuations. Also, if one regards an algorithm as the **while** command

$$\textbf{while } e \notin T \textbf{ do } (e, x) := \gamma(e, x)\,,$$

where the variables e and x range over labels and states, then the tail function is just the direct semantics of this command.

Thus it is hard to say whether Mazurkiewicz's work was a discovery of continuations or a precursor. What is clear is that it inspired the later work of Wadsworth. (See Section 5.)

In 1972, Mazurkiewicz published two further papers [19, 20] concerned with the technique of tail functions.

4. F. L. Morris

In November 1970, F. Lockwood Morris (then a graduate student at Stanford University who was teaching at the University of Essex while working on his dissertation) gave a colloquium at Queen Mary College, London, entitled "The Next 700 Formal Language Descriptions" (an allusion to "The Next 700 Programming Languages" [17], by Peter Landin, who had invited Morris to give the colloquium). In the talk, Morris gave a variety of definitional interpreters for a call-by-value functional language, with extensions to include assignments and labels.

The notes distributed at the talk are published in the present issue of this journal [26]. In the final interpreter in these notes, continuations are used to treat labels and jumps of the kind found in Gedanken [35]. Specifically, what Morris calls *dumps* (since they are abstractions of Landin's dumps) are expression continuations, and what he calls *label values* are command continuations.

Like the SECD machine, but unlike the "circular" interpreters that Morris calls $eval'$, $eval''$, and $eval'''$, the interpreter using continuations defines a call-by-value language regardless of whether the language it is written in uses call by value or call by name.

Looking back on his work, Morris comments:

> I think my main inspiration for programming with continuations was three functions described in the Lisp 1.5 Programmer's Manual [21]: `prop`, `sassoc`, and `search` ... . I can't remember seeing McCarthy or anyone else do the same sort of thing in programming examples, but it may have happened. [25]

He believes he was inspired to a lesser extent by the failure mechanisms in Snobol [11] and Cogent [34]. Before discovering continuations, he had been working on a definition of Snobol that

> ... was as I recall just a working out of the idea that every function needed two continuation arguments, one in case of success and one in case of failure. I think the choice of two continuations was easier to recognize than just one. [25]

The present author had the good fortune to attend Morris's talk. It was my first exposure to the use of continuations, and to the fact that there are many styles of definitional interpreters, varying in abstractness and degree of circularity. These ideas were the genesis of my own work on definitional interpreters [36], which eventually did much to popularize continuations.

5. C. P. Wadsworth

The next bit of our story is most simply told in the first person: While visiting Edinburgh University in December 1970, I had a conversation with Rod Burstall and Chris Wadsworth (then a graduate student at Oxford University) in which I summarized Lockwood Morris's colloquium at Queen Mary. As soon as he grasped the nature of Morris's ideas, Wadsworth exclaimed "That's what I've been working on".

In fact, Wadsworth had discovered the use of continuations to describe the behavior of labels and **goto**'s, and had soon realized that the method also sufficed to describe call by value and other constructs that constrain the order of evaluation. But instead of working with definitional interpreters, he was working with denotational definitions in Dana Scott's then-new lattice-theoretic semantics [39, 40]. In his words:

> Through 1969–70 I had been working on trying to get a good denotational semantics (mathematical semantics as it was then called) [for jumps], under Christopher Strachey's supervision. We devised various "elaborate" schemes, none of which struck us as being at all satisfactory.

> Then I came across a paper by A. Mazurkiewicz [18] ... which gave me the spark we needed. I coined the term "continuation" for the concept/approach that resulted. [47]

Also:

> The other spark was the struggles over earlier months (1969–70) which had got me into "thinking backward" without then seeing how to express it denotationally. Reading Mazurkiewicz's paper gave me the key insight that turned something difficult and messy into something so simple! That sudden insight was: introduce a concept for "the meaning of the rest of the program". [48]

and on the word "continuations":

> Often getting the right word is the catalyst, and so it was for me. Once I'd coined the word it all clicked and the rest (semantic domains, semantic equations, etc.) followed in a matter of a few days. Having the word made it easy and natural to write and discuss, and disseminate (though I thought several times that I would have liked a shorter word!). [48]

Wadsworth delayed publishing his work on continuations, while pursuing his dissertation topics [46] of denotational models of the lambda calculus and graph reduction.

> Apart from the usual reasons things get delayed ..., Strachey felt that it was often good to live with, and try out, a promising idea for a while before publishing — not the dominant practice nowadays (or then?)! He felt, I think, that an idea's originator should allow himself some time to check it out and get it reasonably polished before bothering the world with results that may be of transient value. [48]

Eventually, Strachey described the ideas in a seminar at the Institut de Recherche d'Informatique et d'Automatique in May 1973; a written version was published by IRIA [42], and a revised and slightly expanded version appeared as a report of the Oxford Programming Research Group [43]. Specifically, these reports gave a continuation-style denotational definition of an imperative language with labels and jumps, including jumps out of blocks embedded within expressions. The texts were mostly written by Strachey, but the underlying "method of continuations" was due to Wadsworth. (Although the illustrative language did not include procedures, it appears that Wadsworth understood the treatment of function procedures).

6. J. H. Morris

In the first half of 1971, James H. Morris, Jr. (then at the University of California at Berkeley — and a distant cousin of F. L. Morris) submitted a paper to the Communications of the ACM in which he described a CPS transformation for programs in an Algol-like language (specifically a substantial subset of Algol 60) [28]. The transformation was similar to that of van Wijngaarden, except that it was described in more detail, it avoided the erroneous treatment of nested blocks, and it treated function and proper procedures on the same footing rather than converting function procedures to proper procedures in a preliminary step.

Morris also pointed out that, in addition to eliminating labels and **goto**'s, the transformation would eliminate occurrences of procedures that returned complex results such as arrays, procedures, or labels (assuming it were applied to a richer language that permitted such procedures).

A referee recognized that the basic idea of the paper had been anticipated by van Wijngaarden, and as a consequence the paper was rejected. In its place Morris published a brief letter to the editor [29] that did not describe the program transformation in detail, but demonstrated that it could be used to eliminate procedures returning complex values. (The letter also noted that a conventional stack-based implementation of the transformed program would quickly exhaust its stack.)

In retrospect, Morris comments:

> I simply can't remember the detailed process by which I first discovered the continuation method. However, I originally conceived it as a pure lambda-calculus technique, undoubtedly after several years of living with Peter Landin's J-operator (which must have set the stage for other discoverers, too). I laboriously translated the idea into Algol 60 so as to make the idea more accessible to readers. [30]

7. M. J. Fischer

At the ACM Conference on Proving Assertions about Programs, in Las Cruces, New Mexico in January 1972, Michael J. Fischer (then at MIT) gave a paper entitled "Lambda Calculus Schemata" [12]. (A final, more complete version appears in the present issue of this journal [13].) In this paper, he extended the call-by-value lambda calculus with conditional expressions, and uninterpreted constants and primitive functions; and he described a transformation of this functional language into continuation-passing style.

Fischer's purpose was to show that an arbitrary program can be transformed (by the CPS transformation) into a form that can be implemented by a stack, i.e. where the storage allocated during the execution of a procedure can be deleted when the procedure exits. Of course, the price (as noted by J. H. Morris [29]) is that the stack never pops until the end of program execution.

Fischer's paper is notable for the first proof about the semantics of continuations, i.e. that the CPS transformation preserves meaning in an appropriate sense. This result was in a setting where lambda expressions denote closures. (Results about the meaning of continuations in a denotational-semantics setting, where lambda expressions denote continuous functions, still lay several years in the future [23, 37, 24, 38].)

8. S. K. Abdali

With the appearance of papers by Fischer [12] in January, and J. H. Morris [29] and Reynolds [36] in August, continuations became widely known by the end of 1972. Nevertheless, there was at least one later discovery.

In February 1973, S. Kamal Abdali (then a graduate student at the University of Wisconsin, teaching at New York University while working on his dissertation) presented a short paper at the first Computer Science Conference, held in Columbus, Ohio. In this presentation, Abdali described a novel form of language definition, in which Algol 60 programs were translated into the untyped lambda calculus. This work was then submitted to the first ACM Symposium on Principles of Programming Languages, but was rejected because the extended abstract did not describe any treatment of procedures. Later that summer the work (including a treatment of procedures) was published as a preliminary report [1]. Abdali then moved to Rensselaer Polytechnic Institute, in Troy, New York, where he completed his Wisconsin dissertation [2] in 1974.

According to Abdali, J. Barkley Rosser and his followers (including his student George Petznick, who was Abdali's Ph.D. advisor)

> ... felt that the extensions to the lambda calculus, to which Landin had resorted [15] in establishing a correspondence between Algol 60 and that calculus, made it difficult to use the correspondence for deriving properties of programs. My task, therefore, was to translate programming constructs into the "pure" lambda calclulus. Assignment, for example, was to be modeled by substitution, avoiding the notion of memory, address, and fetch and store operations. [5]

To treat the imperative aspects of Algol 60, Abdali devised a translation much like the CPS transformation. He says that the idea of a continuation (which he called "program remainder")

> ... was inspired by J. H. Morris's thesis [27], in particular, by attempting to get the outline on pp. 38–39 actually to work. Continuations then opened the path to deal with block structure, as well as jumps and labels. The power and significance of continuations was confirmed in overcoming the difficulties of call-by-name; that construct was explicated with immediate and remote continuations to denote calling and called contexts. [5]

Abdali first connected his program remainders with earlier discoveries of continuations (by F. L. Morris and Wadsworth) in the published papers arising from his dissertation [3, 4]. He later used his approach to modeling Algol-like languages in joint work with Franz Winkler [6] and David S. Wise [7].

9. Conclusion

In summary, to the best of this author's knowledge, continuations or closely related concepts were first discovered in 1964 by van Wijngaarden, repeatedly rediscovered in a wide variety of settings — both intellectual and geographical — during 1970 and 1971, and occasionally rediscovered thereafter.

The main mystery is why van Wijngaarden's early work failed to become widely understood. One can speculate, but it is unlikely ever to be known with certainty, particularly since the deaths of Strachey (May 18, 1975) and van Wijngaarden (February 7, 1987).

Nevertheless, the early history of continuations is a sharp reminder that original ideas are rarely born in full generality, and that their communication is not always a simple or straightforward task.

Acknowledgements

For comments and reminiscences that have vastly improved this paper, the author wishes to thank Kamal Abdali, Jaco de Bakker, Olivier Danvy, Edsger Dijkstra, Matthias Felleisen, Andrzej Filinski, Michael Fischer, Dan Friedman, Peter Landin, Antoni Mazurkiewicz, John McCarthy, Douglas McIlroy, Lockwood Morris, James Morris, Peter Naur, Dana Scott, Tom Steel, Guy Steele, Carolyn Talcott, Mads Tofte, and Chris Wadsworth.

References

1. Abdali, S. Kamal. *A Simple Lambda-Calculus Model of Programming Languages*. AEC R & D Report C00–3077–28, New York University (1973).

2. Abdali, S. Kamal. *A Combinatory Logic Model of Programming Languages*. PhD thesis, University of Wisconsin (1974).

3. Abdali, S. Kamal. A lambda-calculus model of programming languages — I. simple constructs. *Journal of Computer Languages*, 1 (1976) 287–301.

4. Abdali, S. Kamal. A lambda-calculus model of programming languages — II. jumps and procedures. *Journal of Computer Languages*, 1 (1976) 303–320.

5. Abdali, S. Kamal. Electronic mail to J. C. Reynolds. (July 21, 1993).

6. Abdali, S. Kamal and Winkler, Franz. *A Lambda-Calculus Model for Generating Verification Conditions*. Technical Report CS–8104, Rensselaer Polytechnic Institute (June 1981).

7. Abdali, S. Kamal and Wise, David S. Standard, storeless semantics for ALGOL-style block structure and call-by-name. In Melton, Austin, editor, *Mathematical Foundations of Programming Semantics*, Springer-Verlag, Berlin (1986) 1–19.

8. Clinger, William, Friedman, Daniel P., and Wand, Mitchell. A scheme for a higher-level semantic algebra. In Nivat, Maurice and Reynolds, John C., editors, *Algebraic Methods in Semantics*, Cambridge University Press, Cambridge, England (1985) 237–250.

9. Dijkstra, Edsger W. Recursive programming. *Numerische Mathematik*, 2 (1960) 312–318.

10. Dijkstra, Edsger W. Go To statement considered harmful. *Communications of the ACM*, 11, 3 (March 1968) 147–148. Letter to the editor.

11. Farber, David J., Griswold, Ralph E., and Polonsky, Ivan P. SNOBOL, a string manipulation language. *Journal of the ACM*, 11, 1 (January 1964) 21–30.

12. Fischer, Michael J. Lambda calculus schemata. In *Proceedings of an ACM Conference on Proving Assertions about Programs* (1972) 104–109.

13. Fischer, Michael J. Lambda-calculus schemata. *Lisp and Symbolic Computation*, 6, 3/4 (1993) 257–286. Appears in this issue.

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

15. Landin, Peter J. A correspondence between ALGOL 60 and Church's lambda-notation. *Communications of the ACM*, 8, 2–3 (February–March 1965) 89–101 and 158–165.

16. Landin, Peter J. *A Generalization of Jumps and Labels*. Report, UNIVAC Systems Programming Research (August 29, 1965).

17. Landin, Peter J. The next 700 programming languages. *Communications of the ACM*, 9, 3 (March 1966) 157–166.

18. Mazurkiewicz, Antoni W. Proving algorithms by tail functions. *Information and Control*, 18, 3 (April 1971) 220–226.

19. Mazurkiewicz, Antoni W. Iteratively computable relations. *Bulletin de l'Académie Polonaise des Sciences Série des Sciences Mathématiques, Astronomiques et Physiques*, 20, 9 (1972) 793–798.

20. Mazurkiewicz, Antoni W. Recursive algorithms and formal languages. *Bulletin de l'Académie Polonaise des Sciences Série des Sciences Mathématiques, Astronomiques et Physiques*, 20, 9 (1972) 799–803.

21. McCarthy, John *et al.* *LISP 1.5 Programmer's Manual*. MIT Press, Cambridge, Massachusetts (1962).

22. McIlroy, M. Douglas. Electronic mail to J. C. Reynolds. (July 14, 1993).

23. Milne, Robert. *The Formal Semantics of Computer Languages and their Implementations*. PhD thesis, Oxford University (1974). Report PRG–13 and Technical Microfiche TCF–2.

24. Milne, Robert and Strachey, Christopher. *A Theory of Programming Language Semantics*. Chapman and Hall, London (1976). In two volumes. Also published by John Wiley, New York.

25. Morris, F. Lockwood. Electronic mail to J. C. Reynolds. (July 13, 1993).

26. Morris, F. Lockwood. The next 700 formal language descriptions. *Lisp and Symbolic Computation*, 6, 3/4 (1993) 249–256. Appears in this issue. Original manuscript dated November 1970.

27. Morris, Jr., James H. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT (December 1968). Report No. MAC–TR–57.

28. Morris, Jr., James H. Eliminating non-local transfers from ALGOL-like languages. (1971). Unpublished.

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

30. Morris, Jr., James H. Electronic mail to J. C. Reynolds. (July 22, 1993).

31. Naur, Peter. The design of the GIER ALGOL compiler, part I. *BIT*, 3 (1963) 124–140. Reprinted in Goodman, Richard, editor, *Annual Review in Automatic Programming*, Vol. 4, Pergamon Press, Oxford (1964) 49–85.

32. Naur, Peter *et al.* Report on the algorithmic language ALGOL 60. *Communications of the ACM*, 3, 5 (May 1960) 299–314.

33. Naur, Peter *et al.* Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 6, 1 (January 1963) 1–17.

34. Reynolds, John C. An introduction to the COGENT programming system. In *Association for Computing Machinery Proceedings of the 20th National Conference* (1965) 422–436.

35. Reynolds, John C. GEDANKEN – a simple typeless language based on the principle of completeness and the reference concept. *Communications of the ACM*, 13, 5 (May 1970) 308–319.

36. Reynolds, John C. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM Annual Conference* (1972) 717–740.

37. Reynolds, John C. On the relation between direct and continuation semantics. In Loeckx, Jacques, editor, *Automata, Languages and Programming*, Springer-Verlag, Berlin (1974) 141–156.

38. Reynolds, John C. Semantics of the domain of flow diagrams. *Journal of the ACM*, 24, 3 (July 1977) 484–503.

39. Scott, Dana S. Outline of a mathematical theory of computation. In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems* (1970) 169–176.

40. Scott, Dana S. *Outline of a Mathematical Theory of Computation.* Technical Monograph PRG–2, Oxford University Computing Laboratory (November 1970).

41. Steele Jr., Guy Lewis and Sussman, Gerald Jay. *LAMBDA: The Ultimate Imperative.* AI Memo 353, Massachusetts Institute of Technology (March 10, 1976).

42. Strachey, Christopher. A mathematical semantics which can deal with full jumps. In *Théorie des Algorithmes, des Langages et de la Programmation*, IRIA (INRIA), Rocquencourt, France (1973) 175–191.

43. Strachey, Christopher and Wadsworth, Christopher P. *Continuations, A Mathematical Semantics for Handling Full Jumps.* Technical Monograph PRG–11, Oxford University Computing Laboratory (January 1974).

44. Sussman, Gerald Jay and Steele Jr., Guy Lewis. *SCHEME: An Interpreter for Extended Lambda Calculus.* AI Memo 349, Massachusetts Institute of Technology (December 1975).

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

46. Wadsworth, Christopher P. *Semantics and Pragmatics of the Lambda-Calculus.* PhD thesis, Oxford University (September 1971).

47. Wadsworth, Christopher P. Electronic mail to Amr A. Sabry. (December 24, 1992).

48. Wadsworth, Christopher P. Electronic mail to J. C. Reynolds. (July 22, 1993).