

Abstract Models of Storage *

ROBERT D. TENNENT DAN R. GHICA {rdt,ghica}@cs.queensu.ca
Computing and Information Science, Queen's University, Kingston, Canada

Abstract. This note is a historical survey of Christopher Strachey's influence on the development of semantic models of assignment and storage management in procedural languages.

Keywords: assignment, storage allocation, locality, parametricity, irreversibility.

1. Introduction

During 1971–73, Christopher Strachey planned to write a major paper entitled *An abstract model of storage*. It is cited as being “in preparation” in two papers [57, 69], and even as “to be published” in a third [59]; however, the projected paper never appeared, and Strachey died in 1975. Nevertheless, the research Strachey and his collaborators did in the years immediately before his death was crucial to the development of semantic models of imperative languages. It has taken researchers some 25 years to make some of those seminal ideas work out, and we still do not have a completely satisfactory theory of storage.

Our aim here is to provide a historical overview of this development. Sections 2–4 outline work immediately before, during, and immediately after Strachey's work in this area, organized historically. Sections 5–9 outline later work, organized around relevant conceptual issues and approaches.

2. Before Strachey

The state of the art in semantic modeling of storage operations *before* Strachey can be seen in two papers by John McCarthy [23, 24]. Commands in simple imperative languages are modeled as transformations of “state vectors” which are intended to record, among other things, the current values of all program variables. Operations for accessing and updating the current value of any variable are axiomatized.

3. Strachey, Burstall, and Park

In a paper presented in 1964 [67], Strachey points out that, in *real* programming languages, left-hand sides of assignments can be complex expressions, and not just variables in the logical sense, and so require a form of evaluation he terms *L*-evaluation, to be distinguished from the conventional *R*-evaluation for the right-hand side expressions of assignments. Furthermore, a variable, in the programmer's

* This work was supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

sense of the word, can have components (arrays, list structures) and can overlap other variables (or components of other variables). Finally, the *number* of variables available to a program execution can change dynamically.

To deal with these complications, Strachey first introduces semantic operators that, when applied to “*L*-values” (generalized addresses) and the current state, can be used to access the current value at that address or to update that value. The essential properties required of an *L*-value are that it must be possible to extract the *R*-value stored at that “area of the store”, and that it must be possible, given a suitable *R*-value, to put this given value in that area to replace the previous occupant. These are termed the *loading* and *updating* components, respectively. Strachey does not limit his view of *L*-values to “basic” (addressable) locations. Note the similarity of this model to the modern object-oriented view of variables [51] as consisting of expression and acceptor components. Load-update pairs soon appeared in programming languages: in CPL [65], as “doublets” in POP-2 [9], and as “implicit references” in GEDANKEN [47].

To deal with dynamic allocation, Strachey introduces operators to allocate and de-allocate *L*-values, and states an axiom intended to ensure that a “new” *L*-value does not overlap any of the previously-existing ones. But this is the only statement of properties expected of the intuitive storage model.

It seems that Strachey first used the term “location” in his 1967 lectures [66], but in a more general sense than in later work on denotational semantic models. In Section 2.2, he emphasizes that a location need not be addressable in general, and can be expressed as a load-update pair. Some examples of the utility of this generalized form of “location” are given in Section 4.1, where he also points out that the components of a load-update pair should satisfy what is now termed the “good-variable” property: the contents of a variable immediately after updating a variable should be the value just assigned.

In Section 3.3.2 of this same paper, however, the “abstract store” is modeled concretely as a *mapping* from *L*-values to *R*-values, with the updating operation defined by the familiar “perturbation” of the store function, involving a test of *L*-value equality. In his appreciation of Strachey and his work [58], Dana Scott quotes an unpublished draft by Robert Milne and Strachey as attributing this idea to a suggestion made to Strachey by Rod Burstall in 1964; however, Burstall himself recalls¹ making the suggestion in the discussion after one of Strachey’s talks in Copenhagen in 1967. The material in Section 3.3.2 of Strachey’s draft might have been prepared *after* his lectures. A work by Burstall on the semantics of assignment was eventually published² in 1968 [8].

At about the same time, David Park [41] was using Strachey’s ideas to analyze the semantics of data structures. The analysis in Section 2 is based on several “basic propositions” (informally stated axioms) about a small variant of Strachey’s model of storage and assignment. Modern readers will detect precursors of several important notions here, including non-interference and good-variables. Furthermore, this work is one of the sources of the important idea of partitioning “computational states” into environment and store components:

The state $\langle \mathcal{L}, \mathcal{C} \rangle$ consists of two mappings \mathcal{L} and \mathcal{C} ; \mathcal{L} is a map from a certain class of expressions, the class of *legal expressions* at that point, into a class of *locations*; \mathcal{C} is a mapping from the class of locations into a class of *objects* (which need not be disjoint from the class of locations). The composition of \mathcal{C} and \mathcal{L} is denoted by \mathcal{R} .

It seems that both the Vienna group [21] and the designers of ALGOL 68 [72] independently developed essentially the same approach at about the same time.

Park's approach to characterizing command effects was to "relate certain assertions about the state at a given moment to assertions about states resulting from commands which might be executed immediately afterwards." It seems that this now-familiar approach was arrived at independently of both Floyd [13] and Hoare [20]. Park concludes this section with the remark that "the notion of 'location' requires further assumptions to be made in order to obtain a cleaner theory."

In Section 3, he changes tack and demonstrates that many language features can be treated as load-update pairs; however, he notes that load-update pairs may not satisfy the "characteristic property" of basic locations that, immediately after an update, the contents is the value just assigned.

4. Strachey, Scott, and Milne

At about this time, Scott began collaborating with Strachey and subsequent events are well-known. An attempt was made to construct a concrete model of the "abstract" store; but Scott noticed that even the simplest model $S = L \rightarrow V$ is problematical when V includes command meanings (in $S \rightarrow S$) or procedures (in $V \times S \rightarrow S$). This forced the initially skeptical Scott [55] to develop domain theory and the first mathematical models of the untyped lambda calculus [56, 59].

But domain theory did not by itself solve all the problems of modeling storage! The most complete presentation of Scott and Strachey's ideas at this time can be found in a 1972 paper by Scott [57]. The main store operators here are as follows:

$$\text{Contents}: L \rightarrow [S \rightarrow V]$$

$$\text{Update}: L \times V \rightarrow [S \rightarrow S]$$

$$\text{Area}: L \rightarrow [S \rightarrow T]$$

$$\text{New}: S \rightarrow L$$

$$\text{Lose}: L \rightarrow [S \rightarrow S]$$

where L is a (flat) domain of *locations*, V is the domain of *storable values*, T is the domain of truth values, and S is "roughly" $A \times [L \rightarrow V]$, where A represents the current active "area." Scott also lists some *postulates*:

$$\text{Contents}(\alpha)(\text{Update}(\alpha, \beta)(\sigma)) = \beta$$

$$\text{Area}(\alpha)(\text{Update}(\alpha, \beta)(\sigma)) = \text{true}$$

$$\text{Contents}(\alpha')(\text{Update}(\alpha, \beta)(\sigma)) = \text{Contents}(\alpha')(\sigma), \text{ when } \alpha' \neq \alpha.$$

But then he states that there is not enough room to formulate all of the necessary equations, and cites Strachey’s *An abstract model of storage* as a report that would give complete details; as already mentioned, this did not work out.

What went wrong? One problem, hinted at in the concluding section of Scott and Strachey’s 1971 paper [59], is that CPL [4, 65], a programming language whose design Strachey was involved in, allowed recursive procedure declarations with side effects. It is quite difficult to describe this feature correctly; the side effects of the declaration should happen only once, and not every time the procedure is called. Robert Milne addressed this problem as part of his work in modeling and verifying implementation methods [26, 27]. Another difficulty with axioms, also discussed by Milne, is that a naive axiomatization of the storage allocation function $new: S \rightarrow L$ combined with the usual monotonicity requirement constrain it to be independent of current stored values, which seems unnecessarily restrictive. Or perhaps, as Milne suggests, it became obvious that intricate “abstract” axioms for the operations are simply less comprehensible than explicit definitions, except for the storage-allocation operator. Similarly, the Vienna group moved from descriptions that were partially axiomatized [22, 19] to more explicit models [5].

5. Possible Worlds

Subsequent developments showed that there were more serious problems with Strachey’s approach to storage allocation. Like most programming-language designers at the time, he was primarily interested in *generalizing* ALGOL 60 to allow dynamic storage allocation; it seemed simplest to use the same approach for both dynamic and block-determined storage. But John Reynolds [51] argued that this semantic model does not do justice to the stack discipline of ALGOL 60.

Later, it became clear [18, 25] that the interactions between local variables and procedures are not properly captured by Strachey’s model of storage allocation; for example, the following equivalence fails:

$$\text{new } x \text{ in } C \equiv C$$

when x is not free in command C .

To make the stack discipline a primary feature of a semantic description, Frank Oles and Reynolds [51, 38, 39, 40] developed a *possible-worlds* form of semantics in which “store shapes” (the way local variables are allocated on the stack) parameterize the meanings of all the state-dependent types in the language; “expansions” (new variable allocations) induce changes of meaning. Thus, a procedure called at a point of execution where the shape of the store differs from that where the procedure was defined has its meaning appropriately tailored. The changes of meaning are subject to certain uniformity conditions.

The immediate consequence is that locality properties of variables are more faithfully reflected in the semantics. It is possible to verify many program equivalences which exemplify locality, such as the equivalence above.

It is interesting to note that Oles characterized the relation between two sets X and Y representing store shapes by axiomatic constraints on a pair of functions

(ϕ, ρ) in a way that resembles Scott and Strachey’s earlier efforts to axiomatize store behaviour. In his model, $\phi: Y \rightarrow X$ is a “forgetting” function allowing a larger store to serve as a smaller one and $\rho: X \times Y \rightarrow Y$ is a “replacement” function used to overwrite a small store embedded into a larger store. The conditions they are required to satisfy are as follows [38, 40]:

$$\begin{aligned}\phi(\rho(x, y)) &= x, \\ \rho(\phi(y), y) &= y, \\ \rho(x, \rho(x', y)) &= \rho(x, y).\end{aligned}$$

Yet Oles shows that, up to isomorphism, models have the form $Y = X \times V$ for a set V of “local” values, with the forgetting function $\phi(x, v) = x$, and the replacement function $\rho(x', (x, v)) = (x', v)$, and so, again, it is not clear that axiomatization provides any significant benefits.

Possible-world semantics allow for two rather different views of the store [33]. One is the abstract or axiomatic view mentioned above, which allows *any* set (up to some cardinality) as a set of states; the other is the lower-level location-oriented approach. The former presents a generalized, object-oriented view of variables, while the latter considers variables simply as generalized addressable memory locations. Interestingly enough, the location-oriented possible-world models [60, 61] seem to be no less powerful in validating program equivalences and enjoy theoretical properties at least as good as the higher-level models. The two alternative views of state raise significant issues of language design, programming style and theoretical detail, but it is unclear whether they ultimately lead to substantially different semantic properties.

Although originally intended to capture stack-like structure of the store, possible-world semantics were later adapted to languages with dynamically allocated variables [28, 42, 63], and to languages with both local and dynamic variables [15].

6. Parametricity

The interactions between procedures and local or dynamically allocated variables in imperative languages raise some additional important issues not adequately explained by possible-world parameterization alone. One is a certain notion of *representation independence* exemplified by equivalences such as:

$$\mathbf{new } x = 0 \mathbf{ in } P(x := x + 1, x) \equiv \mathbf{new } x = 0 \mathbf{ in } P(x := x + 2, x \div 2)$$

where $P(\mathit{inc}, \mathit{val})$ is a procedure accepting as parameters two operations providing “counter” capabilities: an incrementing command inc and an evaluating expression val . The significance of the equivalence is that any such procedure P cannot differentiate between the two implementations of the abstract “counter” (barring overflow, or other similar low-level issues). Such equivalences fail both in classical Strachey-like models and in the original possible-worlds models.

A solution to this problem can be traced back to Section 3.6.4 of Strachey’s lecture notes [66], where he introduces an important distinction between two types

of polymorphism: *ad hoc* and *parametric*. He explains the difference rather informally: parametric polymorphism is “more regular” than the *ad hoc* variety. But parametric polymorphism is also used later on in Section 4.1 in the definitions of variables as generalized “load-update pairs”; that is, Strachey recognized that the basic store operations are parametrically polymorphic.

The significance of parametric polymorphism and, most importantly, its relationship with representation independence was only later elucidated by Reynolds [53]. Finally, Peter O’Hearn and Tennent [36] and, independently, Kurt Sieber [61], showed that induced changes of meaning are in fact parametrically polymorphic with respect to changes of store shape. The resulting parametric possible-world models allow a much wider range of equivalences to be validated, including the one mentioned above.

Years after this work was done, Reynolds disclosed to O’Hearn that in 1975 he had worked out a semantic description of ALGOL 60 in the polymorphic lambda calculus [48]. Reynolds didn’t attempt to publish it because “at the time, this seemed to be a quixotic effort to define a well-understood language in terms of a less understood one” [32].

The exact relationships between parametricity and the uniformity conditions used in possible-world semantics are not yet clear. Possible-world semantics was first presented by Oles and Reynolds using category theory, because some of the desired uniformity conditions were found to correspond to the categorical notion of *naturality*. On the other hand, we do not currently have a general categorical understanding of parametricity [3, 43, 12]. Some researchers [45, 32] believe that naturality will prove to be a special case of parametricity, when the latter is finally understood categorically.

7. Non-Interference and Passivity

David Park had noted [41] that it is virtually impossible to reason about programs in procedural languages without assuming some of the properties of *simple* imperative languages: expressions without side effects and disjointness of variables. These considerations became paramount for the programming-methodology school [17], which rejected the lambda-calculus based approach espoused by Strachey and his collaborators.

John Reynolds has attempted to reconcile these two approaches. In a 1978 paper [49], he shows that *surreptitious* interference can be eliminated in an ALGOL-like language by adopting simple syntactic restrictions which ensure that a procedure and its argument use *disjoint* parts of the state, and that these restrictions can be weakened by taking advantage of the fact that certain types (such as side-effect free expressions) use the state in a strictly “read-only” or *passive* way. Reynolds also showed [50, 52] how to extend Hoare’s logic [20] to full ALGOL-like languages by allowing formal reasoning about non-interference properties. Several researchers have been kept occupied for many years [70, 54, 71, 35, 30, 31, 14] addressing the syntactic, semantic, and even categorical challenges raised by this work.

8. Irreversibility

In 1973, Strachey [69] pointed out a fundamental property of imperative programming that, it seems, had not previously been noted:

The state transformation produced by obeying a command is essentially irreversible and it is, by the nature of the computers we use, impossible to have more than one version of [the state] available at any one time.

Even in the parametric models discussed above, there are operations that are able to “snap back” to previous states of the store. The following fragment [32] fails to be equivalent to $P(\mathbf{diverge})$ if P can snap back to the state with $x = 0$ after executing its argument:

new $x = 0$ **in** $P(x := x + 1)$; **if** $x > 0$ **then** **diverge**

In Sieber’s work [61], snap-backs are accepted as part of the language in order to achieve technical abstractness results; but such languages, while certainly feasible [44, 29], do not benefit from the simple and efficient implementations normally associated with imperative languages.

Irreversibility in procedural languages has recently been addressed by O’Hearn and Reynolds [32] by giving translations into the *polymorphic linear λ -calculus*, a purely functional language with a type structure able to capture both representation independence and irreversibility.

9. Stateless Models

The semantic problems raised by storage are attacked from an entirely different point of view by recent “behavioural” models based on interacting objects [46] or games [1, 2]. These models of storage are more intensional and operational than the state-oriented models considered so far. They are concerned only with *how* programs behave; stores do not appear explicitly in the semantics because they are not directly observable. It is noteworthy that Park’s 1968 paper [41] foreshadows this approach as follows:

The structures constructed in this way can be regarded as finite directed graphs with processes at nodes communicating with adjacent nodes. This suggests to me that, if one is looking for a ‘most general’ form of data structure, the acceptance of such ‘processes’ as fully manipulable computational objects may be the crucial step.

The two views of storage seem at present to be complementary, as with Gordon’s stateful and behavioural descriptions of sequential machines [16]. The stateful view is conceptually familiar and allows particular equivalences to be validated quite easily; the behavioural view is conceptually more sophisticated but allows proofs of general properties such as full abstraction. It is very likely that eventually the two approaches will be reconciled. We may then finally have abstract models of storage that would have satisfied Christopher Strachey.

Acknowledgments

We are grateful for suggestions by Olivier Danvy, Cliff Jones, Robert Milne, Peter Mosses, Peter O'Hearn and John Reynolds.

Notes

1. Personal communication to the first author, October 4th, 1999.
2. In July 1966, Strachey and his CPL collaborators [10] cited a 1965 technical report on CPL semantics by Burstall [7]; but we have been unable to find a copy of this document and the author himself cannot recall what it contained.

References

1. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for IDEALIZED ALGOL with active expressions. In O'Hearn and Tennent [37], pages 297–329 of Volume 2.
2. S. Abramsky and G. McCusker. Full abstraction for IDEALIZED ALGOL with passive expressions. To appear in *Theoretical Computer Science*, 1999.
3. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(10):35–64, 1990. *Corrigendum* in 71(3):431, 1990.
4. D. W. Barron, J. N. Buxton, D. F. Hartley, E. Nixon, and C. Strachey. The main features of CPL. *The Computer J.*, 6:134–143, 1963.
5. H. Bekiç and K. Walk. Formalization of storage properties. In E. Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*, pages 28–61. Springer-Verlag, Berlin, 1971. Also, pages 56–85 in C. B. Jones, editor, *Programming Languages and their Definition*, volume 177 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (1984).
6. S. Brookes, M. Main, A. Melton, and M. Mislove, editors. *Mathematical Foundations of Programming Semantics, Eleventh Annual Conference*, volume 1 of *Electronic Notes in Theoretical Computer Science*, Tulane University, New Orleans, Louisiana, March 29–April 1 1995. Elsevier Science (<http://www.elsevier.nl>).
7. R. M. Burstall. Some aspects of CPL semantics. Technical Report 3, Experimental Programming Unit, University of Edinburgh, 1965.
8. R. M. Burstall. Semantics of assignment. In Dale and Michie [11], pages 3–30.
9. R. M. Burstall and R. J. Popplestone. POP-2 reference manual. In Dale and Michie [11], pages 205–244.
10. J. N. Buxton, G. F. Coulouris, D. F. Hartley, E. Nixon, D. Park, M. Richards, and C. Strachey (editor). CPL Working Papers. Technical report, University of London Institute of Computer Science and Cambridge University Mathematical Laboratory, July 1966. Privately circulated.
11. E. Dale and D. Michie, editors. *Machine Intelligence*, volume 2. Oliver and Boyd, Edinburgh, and American Elsevier, 1968.
12. M. P. Fiore, A. Jung, E. Moggi, P. W. O'Hearn, J. Riecke, G. Rosolini, and I. Stark. Domains and denotational semantics: history, accomplishments and open problems. *Bulletin of the European Association for Theoretical Computer Science*, (59):227–256, 1996. Also available by anonymous FTP as `tenyears.ps.Z` at <ftp.dcs.qmw.ac.uk/pub/lfp/ohearn/>.
13. R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, Providence, Rhode Island, 1967.
14. P. Freyd, P. W. O'Hearn, A. J. Power, M. Takeyama, R. Street, and R. D. Tennent. Bireflectivity. *Theoretical Computer Science*, 228:5–47, 1999.
15. D. R. Ghica. *Semantics of Dynamic Variables in ALGOL-like Languages*. Master's thesis, Department of Computing and Information Science, Queen's University, Kingston, Canada, 1997.

16. M. J. C. Gordon. The denotational semantics of sequential machines. *Information Processing Letters*, 10(1):1–3, February 1980.
17. D. Gries, editor. *Programming Methodology, A Collection of Articles by IFIP WG 2.3*. Springer-Verlag, New York, 1978.
18. J. Y. Halpern, A. R. Meyer, and B. A. Trakhtenbrot. The semantics of local storage, or what makes the free-list free? (preliminary report). In *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages*, pages 245–257, Salt Lake City, Utah, 1984. ACM, New York.
19. W. Henhagl. A storage model derived from axioms. Technical Report TR 25.100, IBM Laboratory, Vienna, 1969.
20. C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580 and 583, 1969.
21. P. Lucas, P. Lauer, and H. Stigleitner. Method and notation for the formal definition of programming languages. Technical Report TR 25.087, IBM Laboratory, Vienna, 1968.
22. P. Lucas and K. Walk. On the formal description of PL/I. In *Annual Review in Automatic Programming, vol. 6*, pages 105–152. Pergamon Press, London, 1969.
23. J. McCarthy. Towards a mathematical science of computation. In *Information Processing (1962)*, pages 21–28. North Holland, 1963.
24. J. McCarthy. A formal description of a subset of ALGOL. In Steel [64], pages 1–7.
25. A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: preliminary report. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 191–203, San Diego, California, 1988. ACM, New York. Reprinted as Chapter 7 of [37].
26. R. E. Milne. *The Formal Semantics of Computer Languages and their Implementations*. Ph.D. thesis, University of Cambridge, and Technical Microfiche TCF-2, Programming Research Group, University of Oxford, 1974.
27. R. E. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, and Wiley, New York, 1976.
28. E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1990.
29. J. G. Morrisett. Refining first-class stores. In SIPL [62], pages 73–87.
30. P. W. O’Hearn. A model for syntactic control of interference. *Mathematical Structures in Computer Science*, 3(4):435–465, 1993.
31. P. W. O’Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. *Theoretical Computer Science*, 228:175–210, 1999. Preliminary version [6] reprinted as Chapter 18 of [37].
32. P. W. O’Hearn and J. C. Reynolds. From ALGOL to polymorphic linear lambda-calculus. To appear in *J. ACM*. Available by anonymous FTP as `AlgolToPolyLin.ps` at `ftp.dcs.qmw.ac.uk/pub/lfp/ohearn/`.
33. P. W. O’Hearn and R. D. Tennent. Semantics of local variables. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 217–238. Cambridge University Press, Cambridge, England, 1992.
34. P. W. O’Hearn and R. D. Tennent. Relational parametricity and local variables. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 171–184, Charleston, South Carolina, 1993. ACM, New York.
35. P. W. O’Hearn and R. D. Tennent. Semantical analysis of specification logic, 2. *Information and Computation*, 107(1):25–57, 1993. Reprinted as Chapter 14 of [37].
36. P. W. O’Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, May 1995. Preliminary version appeared as [34]. Reprinted as Chapter 16 of [37].
37. P. W. O’Hearn and R. D. Tennent, editors. *ALGOL-like Languages*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1997. Two volumes.
38. F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. Ph.D. thesis, Syracuse University, Syracuse, N.Y., 1982.

39. F. J. Oles. Type algebras, functor categories and block structure. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 543–573. Cambridge University Press, Cambridge, England, 1985.
40. F. J. Oles. Functor categories and store shapes. In O’Hearn and Tennent [37], chapter 11, pages 3–12 of Volume 2.
41. D. Park. Some semantics for data structures. In D. Michie, editor, *Machine Intelligence*, volume 3, pages 351–71. American Elsevier, New York, 1968.
42. A. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What’s new? In A. M. Borzyszkowski and S. Sokolowski, editors, *Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 122–140, Gdansk, Poland, 1993. Springer-Verlag, Berlin.
43. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezen and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375, Utrecht, The Netherlands, March 1993. Springer-Verlag, Berlin.
44. B. Randall. System structure for software fault tolerance. *IEEE Transactions in Software Engineering*, SE-1(2):220–232, June 1975.
45. U. Reddy. When parametricity implies naturality. Available by anonymous FTP as `naturality.ps.Z` at `cs.uiuc.edu/pub/faculty/reddy/papers/`, 1997.
46. U. S. Reddy. Global state considered unnecessary: Introduction to object-based semantics. *LISP and Symbolic Computation*, 9(1):7–76, 1996. Reprinted as Chapter 19 of [37].
47. J. C. Reynolds. GEDANKEN—a simple typeless language based on the principle of completeness and the reference concept. *Comm. ACM*, 13(5):308–19, May 1970.
48. J. C. Reynolds. A polymorphic model of ALGOL. Unpublished manuscript, 1975.
49. J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, January 1978. ACM, New York. Reprinted as Chapter 10 of [37].
50. J. C. Reynolds. *The Craft of Programming*. Prentice-Hall International, London, 1981.
51. J. C. Reynolds. The essence of ALGOL. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, Proceedings of the International Symposium on Algorithmic Languages, pages 345–372, Amsterdam, October 1981. North-Holland, Amsterdam. Reprinted as Chapter 3 of [37].
52. J. C. Reynolds. IDEALIZED ALGOL and its specification logic. In D. Néel, editor, *Tools and Notions for Program Construction*, pages 121–161, Nice, France, December 1981. Cambridge University Press, Cambridge, 1982. Reprinted as Chapter 6 of [37].
53. J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Paris, France, 1983. North-Holland, Amsterdam.
54. J. C. Reynolds. Syntactic control of interference, part 2. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 704–722, Stresa, Italy, July 1989. Springer-Verlag, Berlin.
55. D. S. Scott. A type-theoretical alternative to CUCH, ISWIM, OWHY. Privately circulated memo, Oxford University, October 1969. Published in *Theoretical Computer Science*, 121(1/2):411–440, 1993.
56. D. S. Scott. Outline of a mathematical theory of computation. In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176, Princeton, 1970. Department of Electrical Engineering, Princeton University. Also Technical Monograph PRG-2, Oxford University Computing Laboratory, Programming Research Group, Oxford.
57. D. S. Scott. Mathematical concepts in programming language semantics. In *Proc. 1972 Spring Joint Computer Conference*, pages 225–34. AFIPS Press, Montvale, N.J., 1972.
58. D. S. Scott. An appreciation of Christopher Strachey and his work. Foreword to J. E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, pages xv–xxx. The MIT Press, Cambridge, Massachusetts, and London, England, 1977.
59. D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of the Symposium on Computers and Automata*, volume 21 of *Microwave Research Institute Symposia Series*, pages 19–46. Polytechnic Institute of Brooklyn

- Press, New York, 1971. Also Technical Monograph PRG-6, Oxford University Computing Laboratory, Programming Research Group, Oxford.
60. K. Sieber. New steps towards full abstraction for local variables. In SIPL [62], pages 88–100.
 61. K. Sieber. Full abstraction for the second order subset of an ALGOL-like language. In *Mathematical Foundations of Computer Science*, volume 841 of *Lecture Notes in Computer Science*, pages 608–617, Kőšice, Slovakia, August 1994. Springer-Verlag, Berlin. Reprinted as Chapter 15 of [37].
 62. *ACM SIGPLAN Workshop on State in Programming Languages*, Copenhagen, Denmark, June 12, 1993. Technical report RR-968, Department of Computer Science, Yale University.
 63. I. Stark. Categorical models for local names. *LISP and Symbolic Computation*, 9(1):77–107, February 1996.
 64. T. B. Steel, Jr., editor. *Formal Language Description Languages for Computer Programming, Proceedings of the IFIP Working Conference*, Baden bei Wien, Austria, September 1964. North-Holland, Amsterdam (1966).
 65. C. Strachey. *CPL Reference Manual (Incomplete Draft)*. In [10].
 66. C. Strachey. *Fundamental Concepts in Programming Languages*. Unpublished lecture notes, International Summer School in Computer Programming, Copenhagen, August 1967. Reprinted in this volume.
 67. C. Strachey. Towards a formal semantics. In Steel [64], pages 198–220.
 68. C. Strachey. The varieties of programming language. In *Proceedings of the International Computing Symposium*, pages 222–233, Venice, April 1972. Cini Foundation, Venice.
 69. C. Strachey. The varieties of programming language. Technical monograph PRG-10, Oxford University Computing Laboratory, Programming Research Group, Oxford, March 1973. Revised and slightly expanded version of [68]. Reprinted as Chapter 2 of [37].
 70. R. D. Tennent. Semantics of interference control. *Theoretical Computer Science*, 27:297–310, 1983.
 71. R. D. Tennent. Semantical analysis of specification logic. *Information and Computation*, 85(2):135–162, 1990. Reprinted as Chapter 13 of [37].
 72. A. van Wijngaarden (ed.) et al. Report on the algorithmic language ALGOL 68. *Numerische Mathematik*, 14:79–218, 1969.