8-1994

Introduction to the Literature on Semantics

Gary T. Leavens
Iowa State University

Introduction to the Literature On Semantics

Gary T. Leavens

TR #94-15
August 1994

Keywords: programming languages, axiomatic sematics, denotational semantics, operational semantics, type systems, polymorphism, type theory, data abstraction.
1992 CR Categories: D.2.4 [*Software Engineering*] Program Verification; D.3.1 [*Programming Languages*] Formal Definitions and Theory; F.3.1 [*Logics and Meaning of Programs*] Specifying and verifying and reasoning about programs; F.3.2 [*Logics and Meaning of Programs*] Semantics of Programming Languages; F.3.3 [*Logics and Meaning of Programs*] Studies of Program Constructs.

Department of Computer Science
226 Atanasoff Hall
Iowa Sate University
Ames, Iowa 50011-1040, USA

Introduction to the Literature on Semantics

Gary T. Leavens
Department of Computer Science, Iowa State University
Ames, IA, 50011-1040 USA
leavens@cs.iastate.edu

August 16, 1994

Abstract

An introduction to the literature on semantics. Included are pointers to the literature on axiomatic semantics, denotational semantics, operational semantics, and type theory.

The following is an selective introduction to the literature on programming language semantics. Since this is an *introduction*, only a small fraction of the literature can be mentioned here. References are included because they have intrinsic interest, although some are included simply because they are the original sources and are likely to be referenced by others doing related work (e.g. [Chu41]). If you wish to probe an area more deeply, you might start with the papers mentioned, follow their references, and also use the *Science Citation Index* to see what papers have referenced the ones mentioned.

1 General Sources

Two recent texts that cover many different approaches to semantics are: [NN92] [Win93].

Articles discussing principal topics in semantics, containing many references, appear in volume B of the *Handbook of Theoretical Computer Science* [vL90].

Journals that include a substantial coverage of semantics include *ACM Transactions on Programming Languages and Systems* (TOPLAS), *ACM SIGPLAN Notices*, *IEEE Transactions on Software Engineering*, *Information and Computation* (formerly *Information and Control*, Academic Press), and *Acta Informatica* (Springer-Verlag). Related areas, such as specification and verification have their own journals (e.g., *ACM Transactions on Software Engineering and Methodology*, *Science of Computer Programming*, and *Formal Aspects of Computing*).

Conferences devoted to topics related to semantics include the IEEE Annual Symposium on Logic in Computer Science (LICS), the Annual ACM Symposium on Principles of Programming Languages (POPL), Mathematical Foundations of Programming Semantics (MFPS), the European Symposium on Programming (ESOP), the ACM Symposium on LISP and Functional Programming (LFP), the International Colloquium on Automata, Languages, and Programming (ICALP), and the International Symposium on Mathematical Foundations of Computer Science. Conferences not sponsored by the ACM or IEEE are often available in the Springer-Verlag series of Lecture Notes in Computer Science (LNCS), and workshops are often available in the series Workshops in Computing. The above list does not include conferences devoted to software engineering (e.g. TAPSOFT) or particular methods (e.g., category theory) where some important work is also found.

2 Mathematical Background

The basics of universal algebra underly much of the work in semantics. Gratzer's monumental book [Gra79] is a standard reference. A generalized notion of homomorphism that is often used in semantics is Statman's logical relations [Sta85], explained in [Mit90].

Another foundation is the area of mathematical logic. A classic approach, using proof theory and model theory is [End72]. More recent approaches, favoring proof theory (especially predicate logic) at the expense of model theory and mathematical depth and emphasizing calculation, include [DS90] [GS94] [MW93].

2.1 λ-Calculus

A standard reference on Church's (untyped) λ-calculus [Chu41] is Barendregt's book [Bar84]. This book is at times highly technical, but also has sensible definitions and introductory material. A shorter account, with a reasonable introduction to the λ-calculus is found in [Bar90]. Other introductions include [Gor88] [Sch94, Sections 6.1–6.2].

The use of λ-calculus for describing programming languages and as the inspiration for programming language design has been investigated by Landin [Lan65] [Lan66] and many others.

For the typed λ-calculus, a standard reference is [GLT89]. A good introduction can be found in [Gun92, Chapter 2] and [Sch94, Sections 6.4]; both of these texts discuss the semantics of the typed lambda calculus in detail. Schmidt's book [Sch94] is also a good introduction to higher-order typed lambda calculi, and contains references to work on such higher-order extensions.

2.2 Category Theory

Increasingly, especially in denotational semantics, category theory [Lan71] [LS91] is used in research and for the elegant presentation of results. For example, categorical logic and the semantics of the typed lambda calculus are discussed in [LS86].

For me, the best introduction seems to be the first 5 chapters of [Gol84]. There are now several articles [Hoa89] and books [BW90] [AL91b] [Pie91] [Wal91] explaining category theory to computer scientists. For those familiar with (or needing) background in the semantics of abstract data type specifications, a good introduction is Burstall and Goguen's paper [BG82].

3 Axiomatic Semantics

Hoare pointed out, in his seminal paper [Hoa69, Section 6], that one could define a programming language by insisting "that all implementations of the language shall 'satisfy' the axioms and rules of inference which underlie proofs of the properties of the programs expressed in the language." (However, Hoare attributes this idea to Floyd [Flo67].) Such a semantics is useful because it directly aims to support program verification.

Because a logic for program verification can be used to define a programming language, it is difficult to disentangle work on program verification from work on semantics. Even if program verification is not itself a form of semantics (rather, it is an application), it still provides important motivation for axiomatic semantics. Therefore, verification is also discussed in this section.

3.1 Program Verification

Program verification is concerned with showing that a particular program meets its specification. This may be accomplished in several ways; for example, reasoning from the denotational semantics of the program. However, the term "program verification" is usually implies a syntactic proof in a specially constructed logic.

Floyd introduced the inductive assertions and well-founded sets methods for program verification in 1967 [Flo67] (see also [LS87, Chapter 8]). (See [Cou90] for historical references to the work of Naur, Von Neumann, and Turing, who anticipated this work.) The term inductive assertions refers to the assertions that are put at every branch point in the program's flow graph; these give rise to verification conditions, which state that along every path, the correctness of the assertions is preserved. This shows partial correctness (that the program is correct if it terminates). The well-founded sets method allows proofs of termination.

Hoare [Hoa69] was the first to present verification in terms of a logic for partial correctness that was compositional, so that program proofs are done by induction on the syntax of programs. This paper sparked a wealth of research in program verification. Some surveys of results of this research are [Apt81], [FB86], and [Cou90], the latter of which also cites other surveys. A technical reference is [LS87, Chapter 9].

A very introductory tutorial on verification from the software engineering perspective is [LG86, chapter 11]. The idea of developing a proof of a program at the same time the program is being developed has been eloquently advocated by Dijkstra and Gries [Dij76] [Gri81]. See [Gor88] for a more theoretical introduction that treats some aspects of theorem proving.

Concurrency has always been a prime application area, because concurrent programs are so difficult to debug. Verification of concurrent programs may be handled with the Owicki-Gries method [OG76]. Other techniques use temporal logic (e.g., [OL82]) and "transition axioms" [Lam89]. An introduction to Hoare-style verification that discusses concurrent and distributed programs is [Al91a].

3.1.1 Abstract Data Types

Hoare also was the first to consider how verification and abstract data types interact [Hoa72]. His technique allows one to separately verify the correctness of a type's implementation and programs that use that type, and is based on the use of representation invariants and abstraction functions. A practical implementation of these ideas is found in [NHN80]. VDM also uses these techniques [Jon90]. More recent work in this area is found in [Sch90] and [Nip89].

Abstract data types (ADTs) can be specified algebraically [GHM78] [EM85] or by using pre- and post-conditions [BJ82] [Jon86] [Win87]. See [LG86, chapter 11] for a tutorial on program verification using abstract data types.

A related aspect is how the structure of a proof of correctness may help follow the hierarchical (i.e., layered) structure of the implementation design [RL77] [GMP90] [Sch82].

3.1.2 Transformational Programming

Hoare's original paper [Hoa69], treated verification as something to be done after a program was written. However, Hoare [Hoa71] and others were soon advocating the development of proofs at the same time as programs. Dijkstra and Gries became prime advocates of this technique [Dij76] [Gri81]. More recent treatments in this style advocate a calculational approach [Coh90] [GS94]. One formalization of this approach is the refinement calculus [Bv89a][Bv89b] [Mor90] [MG90] [MV94].

The idea of formal development of program and proof has found favor in the functional programming community as well. The idea behind transformational programming is to transform the specification of a program into an efficient version [BD77] [MW80] [Bal81] [CHT81]; these references are characterized by their work with equational logic (instead of predicate transformers or a Hoare logic), their use of algebraic techniques, and the general focus on functional languages. Some recent work in this area can be found in [Mee87] [Bir89a] [Bir89b] [Tal88]. Transformational (or algebraic) techniques also underly the work on Backus's FP [Bac78] and Hoare's "Laws of Programming" for CSP [HHJ$^+$87].

Contrary to popular belief, it is also possible to reason "equationally" about programs with side-effects and even continuations [Boe85] [FF86] [FH89].

3.2 Axiomatic Semantics of Programming Languages

A classic text on the semantics of programming languages which treats axiomatic semantics is [dB80a]. Predicte transformer semantics, using the "weakest precondition" operator were advocated by Dijkstra for the definition of his guarded command language [Dij75]. A monograph on predicate transformer semantics is [DS90] which goes over notational issues as well as the underlying mathematics. A recent monograph that updates [dB80a] using predicate transformers is [Hes92].

An example of the use of the axiomatic semantics to describe a programming language is Hoare and Wirth's axiomatic definition of Pascal [HW73].

The use of an axiomatic semantics as a metric for the "goodness" of a language led to two language designs in the 1970s. Although never implemented fully, Alphard [SWL77] [Sha81] was interesting, and focused on support for both data abstraction and verification. The language Euclid [LGH$^+$78] [PHL$^+$77] was implemented, and has been used for various large projects.

3.3 Algebraic Semantics

While most axiomatic approaches use Hoare logic or variations on Hoare logic, there have been a few attempts to use the techniques of algebra and equational specification to define programming languages. Examples include [MA86] [BWP87].

4 Denotational Semantics

In contrast to axiomatic semantics, denotational semantics [Str66] [SS71] [MS76] [Sto77] [Sco81] [Sch86] explicitly constructs mathematical models of programming languages. A short summary of the denotational approach can be found in Tennent's article "The Denotational Semantics of Programming Languages" [Ten76]. A recent survey is found in [Mos90].

Introductory texts include [Gor79], [All86], [Wat91], [Sch94]. An excellent new graduate text with more mathematical depth is [Gun92]. Standard works on denotational semantics are the books by Stoy [Sto77] and Schmidt [Sch86], both of which offer a comprehensive and mathematical treatment. Schmidt's book [Sch86] has an excellent discussion of domain theory (see also [GS90]) and can be consulted for references to denotational descriptions of real languages.

One can use a typed functional programming language, such as Standard ML, to implement a denotational semantics. Two descriptions of this idea are [Wat86] [MA89].

Action semantics, an offshoot of denotational semantics, is described in [Wat91] and more fully in [Mos92].

Schmidt's book [Sch86] also has a discussion of denotational semantics for concurrent systems. Hennessy's book has a discussion of more recent work in this area [Hen88]. However, denotational approaches to concurrency have not been as successful as operational semantics.

5 Operational Semantics

A compiler or interpreter gives an operational semantics to a programming language. This style of semantics dates back to the earliest programming languages, and was first formalized in the idea of a meta-circular interpreter for LISP [McC60].

Meta-circular interpreters are still useful for teaching purposes, and for prototyping programming language designs. Several meta-circular interpreters for variants of LISP are discussed in Steele and Sussman's paper *The Art of the Interpreter* [SS78]. An excellent and more readily accessible discussion is found in Abelson and Sussman's book [ASS85], which uses Scheme. A more detailed treatment of interpreters is found in [Kam90] [FWH92]. See [KdRB91] for an approach using the CLOS meta-object protocol.

However, for mathematical convenience, one wants something more abstract than an interpreter or complier. Landin overcame the circularity problem by the use of an abstract machine called the "SECD machine" [Lan64] (see also [Hen80]).

A more systematic style of operational semantics based on rewrite rules is found in Plotkin's terminal transition systems [Plo77] also known as "structural operational semantics" [Plo81] [Hen90] or a "labeled transition system" [Ast91]. Hennessy's book is an elementary introduction [Hen90]. This style of semantics has the advantage that it extends naturally to studies of concurrency [Mil90]. A classic reference for the operational semantics of concurrent processes is Milner's book on CCS [Mil80]. See [Hen88] and [Mil90] for more recent work in this area.

6 Type Systems

Much of the modern study of semantics concerns type systems, which describe many aspects of the static semantics of programming languages. In contrast, axiomatic, denotational, and operational semantics describe the dynamic semantics of programming languages.

The purpose of type checking is nicely summarized by Morris as a mechanism that allows program modules to protect objects from unwanted discovery, modification, and impersonation [Mor73]. Wegbreit's discussion of the extensible language EL1, is also good background [Weg74].

Schmidt's recent book [Sch94] starts treats type systems in the context of programming language semantics. It also goes deeply into more advanced topics such as higher-order type systems and predicate-logic typing.

6.1 Polymorphism

An introductory survey of modern polymorphic type systems and research results is Cardelli and Wegner's paper "On Understanding Types, Data Abstraction and Polymorphism" [CW85]. See also [Har84] [Car91] [DT88]; the latter two have much material related to object-oriented programming. A still more recent survey is [Mit90].

Standard references include Girard's system Fω [Gir71] (see also [Gir86] [GLT89]), and Reynolds independent work [Rey74], sometimes called the Girard-Reynolds second order lambda calculus (or SOL). Modern expositions are found in [MP85] [MH88] [Rey85] [Mit90] [Car91].

Other kinds of type information may be incorporated into a type system and checked at the same time as types [GL86] [LG88] [OG89].

6.2 Type Theory

Type theory, narrowly defined, uses the tools of constructive logic to study polymorphic type systems. A good introduction can be found in [Sch94, Chapters 8–10]. Logical inference systems can often be translated directly into type systems due to the "Formula as Types" notion or the "Curry-Howard isomorphism" [How80] [GLT89, Chapter 3] [Con89]. Thus much research in type theory lies on the border of mathematics and computer science. Another motivation is to use type information to capture behavioral specifications, thus allowing one to reason about programs in the programming language [NP83] [Dyb90].

The principal groups working on type theory include deBruijn and others working on AUTOMATH [dB80b], Martin-Löf's and followers [ML75] [ML82] [Bac89], Constable's group at Cornell has been very active in this area. Their language PRL which uses constructive mathematics, is described in the paper "Proofs as Programs" [BC85]. A related language is PL/CV3 [CZ84]. Coquand and Huet's group is responsible for the "calculus of constructions" [CH88]. A recent survey with more references is [BH90a].

6.3 Data Abstraction: ADTs and OOP

Reynolds [Rey75] [Rey78] (see also [Coo91]) distinguished two ways in which a type system could support data abstraction. The first way is to have the language give an object different types outside and inside a defining module; this Reynolds called "user defined types". User defined types also go by the name of "abstract data types" (ADTs), and their support in a type system is exemplified by the language CLU [LSAS77] [LG86].

Mitchell and Plotkin connected this approach, ADTs, to the second-order lambda calculus [MP85]. This idea has been used to show some representation independence results [Mit86]. For other work along these lines, see for example, [Mac86] [CL90] [Mit90].

The second approach to supporting data abstraction Reynolds called "procedural data abstraction". Today it goes under the more common name of "object-oriented programming" (OOP) [DMN70] [BDMN73] [GR83]. Some languages exemplifying typed support for OOP include SIMULA [BDMN73], and Eiffel [Mey88] [Mey92], both of which have insecure type systems [Coo89]. The language Trellis/Owl [SCB$^+$86] features by-name type checking and subtyping. By contrast,Emerald [BHJL86] [BHJ$^+$87] [BH90b] [BH91] and Quest [Car91] feature structural subtyping.

Some seminal references on types and OOP are reprinted in [GM94]. Recent theoretical work on types for OOP includes the following [Car88b] [Car88a] [CM89] [AC90] [CMMS91] [Car93] [CCH$^+$89] [CHC90] [Coo89] [BTGS90] [HP90] [MMM91] [BCM$^+$93] [BL90] [BM92] [Aba93] [Bru93]. (Cardelli is one of the most active in this area, and most of the literature will cite one of his papers.) For work that directly bears on multimethods (as in CLOS), see [Rey80] [Ghe91a] [Ghe91b] [CGL92] [Cha92] [CGL93] [Cas93] [CL94].

6.4 Type Reconstruction

Type reconstruction (or type inference) is the process of reconstructing types for a program that has no type declarations. A practical and sound algorithm is described in Milner's paper "A Theory of Type Polymorphism in Programming" [Mil78]. Another exposition of this material, incorporating refinements found in [DM82], is found in [Car87].

Milner-style type inference system is used in the programming language ML [GMW79] [Mil84] [Har86] [HMM86]. The ideal model has been used to give a semantics to the ML type system [MPS86].

The question of whether one can do type inference for more powerful type systems has been an active area of research (e.g., [McC84] [Wan89]). A recent survey of what is known about such problems is found in [Tiu90].

See Knight's survey [Kni89] for a discussion of the central role that unification plays in type inference and more details on unification itself.

6.5 Research Languages

The language Russell was developed at Cornell to investigate how types can be treated as values. There are many papers that have appeared about Russell, but perhaps the best introduction to the language is the paper "Data Types are Values" [DD85], which can be consulted for other references. Russell has separate mechanisms for building types and for information hiding; the same idea is found in Haskell [Hud89, Pages 387-388]. Interesting variations are found in Miranda [Tur85].

Much recent work has centered around the language ML and its modern variant Standard ML [Mil84] [MTH90] [MT91]. Besides the type inference in its type system mentioned above, Standard ML has an interesting module system [Mac84] [Har85] [HMT87].

References

[Aba93] Martin Abadi. Baby Modula-3 and a Theory of Objects. Technical Report 95, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, February 1993. Order from src-report@src.dec.com.

[AC90] Roberto M. Amadio and Luca Cardelli. Subtyping Recursive Types. Technical Report 62, Digital Systems Research Center, Palo Alto, Ca 94301, August 1990. See also the 1991 POPL proceedings.

[Al91a] Krzysztof R. Apt and Ernst-Rudiger 0lderog. Introduction to Program Verification. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 363–429. Springer-Verlag, New York, N.Y., 1991.

[AL91b] Andrea Asperti and Guiseppe Longo. *Categories, Types and Structures.* The MIT Press, Cambridge, Mass, 1991.

[All86] Lloyd Allison. *A Practical Introduction to Denotational Semantics.* Cambridge University Press, New York, N.Y., 1986.

[Apt81] Krzystof R. Apt. Ten Years of Hoare's Logic: A Survey—Part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, October 1981.

[ASS85] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs.* The MIT Press, Cambridge, Mass., 1985.

[Ast91] Edigio Astesiano. Inductive and Operational Semantics. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 51–136. Springer-Verlag, New York, N.Y., 1991.

[Bac78] John Backus. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. *Communications of the ACM*, 21(8):613–641, August 1978.

[Bac89] R. C. Backhouse. Constructive Type Theory – An Introduction. In Manfred Broy, editor, *Constructive Methods in Computing Science*, volume F55 of *NATO ASI Series*, pages 9–60. Springer-Verlag, New York, N.Y., 1989.

[Bal81] Robert Balzer. Transformational Implementation: an Example. *IEEE Transactions on Software Engineering*, SE-7(1), January 1981.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland Publishing Co., New York, N.Y., 1984. Revised Edition.

[Bar90] H. P. Barendregt. Functional Programming and Lambda Calculus. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 7, pages 321–363. The MIT Press, New York, N.Y., 1990.

[BC85] Joseph L. Bates and Robert L. Constable. Proofs as Programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, January 1985.

[BCM$^+$93] Kim B. Bruce, Jon Crabtree, Thomas P. Murtagh, Robert van Gent, Allyn Dimock, and Robert Muller. Safe and decidable type checking in an object-oriented language. *ACM SIGPLAN Notices*, 28(10):29–46, October 1993. *OOPSLA '93 Proceedings*, Andreas Paepcke (editor).

[BD77] R. M. Burstall and J. L. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[BDMN73] Graham M. Birtwistle, Ole-Johan Dahl, Bjorn Myhrhaug, and Kristen Nygaard. *SIMULA Begin.* Auerbach Publishers, Philadelphia, Penn., 1973.

[BG82] R. M. Burstall and J. A. Goguen. Algebras, Theories and Freeness: An Introduction for Computer Scientists. In Manfred Broy and Gunther Schmidt, editors, *Theoretical Foundations of Programming Methodology: Lecture Notes of an International Summer School directed by F. L. Bauer, E. W. Dijkstra and C. A. R. Hoare*, volume 91 of *series C*, pages 329–348. D. Ridel, Dordrecht, Holland, 1982.

[BH90a] Henk Barendregt and Kees Hemerik. Types in Lambda Calculi and Programming Languages. In N. Jones, editor, *ESOP '90 3rd European Symposium on Programming, Copenhagen, Denmark*, volume 432 of *Lecture Notes in Computer Science*, pages 1–35. Springer-Verlag, New York, N.Y., May 1990.

[BH90b] Andrew P. Black and Norman C. Hutchinson. Typechecking Polymorphism in Emerald. Technical Report TR 90-34, Department of Computer Science; The University of Arizona, Tucson, AZ 85721, December 1990.

[BH91] Andrew P. Black and Norman Hutchinson. Typechecking Polymorphism in Emerald. Technical Report CRL 91/1 (Revised), Digital Equipment Corporation, Cambridge Research Lab, Cambridge, Mass., July 1991.

[BHJ$^+$87] Andrew Black, Norman Hutchinson, Eric Jul, Henry Levy, and Larry Carter. Distribution and Abstract Types in Emerald. *IEEE Transactions on Software Engineering*, SE-13(1):65–76, January 1987.

[BHJL86] Andrew Black, Norman Hutchinson, Eric Jul, and Henry Levy. Object Structure in the Emerald System. *ACM SIGPLAN Notices*, 21(11):78–86, November 1986. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

[Bir89a] R. S. Bird. Algebraic Identities for Program Calculation. *The Computer Journal*, 32(2):122–126, April 1989.

[Bir89b] Richard S. Bird. Lectures on Constructive Functional Programming. In Manfred Broy, editor, *Constructive Methods in Computing Science*, volume F55 of *NATO ASI Series*, pages 151–216. Springer-Verlag, New York, N.Y., 1989.

[BJ82] Dines Bjorner and Cliff B. Jones. *Formal Specification and Software Development*. International Series in Computer Science. Prentice-Hall, Inc., London, 1982.

[BL90] K. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990.

[BM92] Kim Bruce and John C. Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–327. ACM, January 1992.

[Boe85] Hans-Juergen Boehm. Side Effects and Aliasing Can Have Simple Axiomatic Descriptions. *ACM Transactions on Programming Languages and Systems*, 7(4):637–655, October 1985.

[Bru93] K. Bruce. Safe Type Checking in a Statically Typed Object-Oriented Programming Language. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 285–298, 1993.

[BTGS90] V. Breazu-Tannen, C. A. Gunter, and A. Scedrov. Computing with Coercions. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice, France*, pages 44–60. ACM, June 1990.

[Bv89a] R. J. R. Back and J. von Wright. Refinement Calculus, Part I: Sequential Nondeterministic Programs. Technical Report Ser. A, No 92, Abo Akademi University, Department of Computer Science, Lemminkäinengatan 14, 20520 Abo, Finland, 1989. Appears in *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop*, Mook, The Netherlands, May/June 1989, Spring-Verlag, LNCS 430, J. W. de Bakker, et al, (eds.), pages 42–66.

[Bv89b] R. J. R. Back and J. von Wright. Refinement Calculus, Part II: Parallel and Reactive Programs. Technical Report Ser. A, No 93(?), Abo Akademi University, Department of Computer Science, Lemminkäinengatan 14, 20520 Abo, Finland, 1989. Appears in *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop*, Mook, The Netherlands, May/June 1989, Spring-Verlag, LNCS 430, J. W. de Bakker, et al, (eds.), pages 67–93.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science*. International Series in Computer Science. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1990. ISBN 0-13-120486-6.

[BWP87] Manfred Broy, Martin Wirsing, and Petter Pepper. On the Algebraic Definition of Programming Languages. *ACM Transactions on Programming Languages and Systems*, 9(1):54–99, January 1987.

[Car87] Luca Cardelli. Basic Polymorphic Typechecking. *Science of Computer Programming*, 8(2), April 1987.

[Car88a] Luca Cardelli. A Semantics of Multiple Inheritance. *Information and Computation*, 76(2/3):138–164, February/March 1988. A revised version of the paper that appeared in the 1984 Semantics of Data Types Symposium, LNCS 173, pages 51–66.

[Car88b] Luca Cardelli. Structural Subtyping and the Notion of Power Type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, Calif.*, pages 70–79. ACM, January 1988.

[Car91] Luca Cardelli. Typeful Programming. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 431–507. Springer-Verlag, New York, N.Y., 1991.

[Car93] Luca Cardelli. An Implementation of $F_{<:}$. Technical Report 97, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, February 1993. Order from src-report@src.dec.com.

[Cas93] G. Castagna. A Meta-Language for Typed Object-Oriented Languages. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, October 1993.

[CCH$^+$89] Peter Canning, William Cook, Walter Hill, John Mitchell, and Walter Olthoff. F-Bounded Polymorphism for Object-Oriented Programming. In *Fourth International Conference on Functional Programming and Computer Architecture*. ACM, September 1989. Also technical report STL-89-5, from Software Technology Laboratory, Hewlett-Packard Laboratories.

[CGL92] Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A Calculus for Overloaded Functions with Subtyping. In *ACM Conference on LISP and Functional Programming*, pages 182–192. ACM, June 1992. To appear in *Information and Computation*.

[CGL93] G. Castagna, G. Ghelli, and G. Longo. A semantics for λ&-*early*: a calculus with overloading and early binding. In M. Bezem and J.F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science, pages 107–123, Utrecht, The Netherlands, March 1993. Springer-Verlag. TLCA'93.

[CH88] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[Cha92] Craig Chambers. Object-Oriented Multi-Methods in Cecil. In Ole Lehrmann Madsen, editor, *ECOOP '92, European Conference on Object-Oriented Programming, Utrecht, The Netherlands*, volume 615 of *Lecture Notes in Computer Science*, pages 33–56. Springer-Verlag, New York, N.Y., 1992.

[CHC90] William R. Cook, Walter L. Hill, and Peter S. Canning. Inheritance is Not Subtyping. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California*, pages 125–135, January 1990. Also STL-89-17, Software Technology Laboratory, Hewlett-Packard Laboratories, Palo Alto, Calif., July 1989.

[CHT81] T. E. Cheatham, G. H. Holloway, and J. A. Townley. Program Refinement by Transformation. In *Fifth International Conference on Software Engineering*, pages 430–437. IEEE, 1981.

[Chu41] A. Church. *The Calculi of Lambda Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, N.J., 1941. Reprinted by Klaus Reprint Corp., New York in 1965.

[CL90] Luca Cardelli and Xavier Leroy. Abstract Types and the Dot Notation. Technical Report 56, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, March 1990. Order from src-report@src.dec.com.

[CL94] Craig Chambers and Gary T. Leavens. Typechecking and Modules for Multi-Method. Technical Report 94-03, Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, Iowa 50011, March 1994. Also University of Washington Department of Computer Science and Engineering TR number 94-03-01. To appear in OOPSLA '94.

[CM89] Luca Cardelli and John C. Mitchell. Operations on Records (Summary). In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University*, volume 442 of *Lecture Notes in Computer Science*, pages 22–52. Springer-Verlag, New York, N.Y., March 1989.

[CMMS91] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An Extension of System F with Subtyping. Technical Report 80, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, December 1991. Extended abstract in Theoretical Aspects of Computer Software, T. Ito and A. R. Meyer (editors), LNCS Vol 526. To appear in Information and Control.

[Coh90] Edward Cohen. *Programming in the 1990s: An Introduction to the Calculation of Programs*. Springer-Verlag, New York, N.Y., 1990.

[Con89] Robert L. Constable. Assigning Meaing to Proofs: a semantic basis for problem solving environments. In Manfred Broy, editor, *Constructive Methods in Computing Science*, volume F55 of *NATO ASI Series*, pages 63–91. Springer-Verlag, New York, N.Y., 1989.

[Coo89] W. R. Cook. A Proposal for Making Eiffel Type-safe. *The Computer Journal*, 32(4):305–311, August 1989.

[Coo91] William R. Cook. Object-Oriented Programming Versus Abstract Data Types. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, New York, N.Y., 1991.

[Cou90] Patrick Cousot. Methods and Logics for Proving Programs. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 15, pages 841–993. The MIT Press, New York, N.Y., 1990.

[CW85] Luca Cardelli and Peter Wegner. On Understanding Types, Data Abstraction and Polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[CZ84] Robert L. Constable and Daniel R. Zlatin. The Type Theory of PL/CV3. *ACM Transactions on Programming Languages and Systems*, 6(1):94–117, January 1984.

[dB80a] Jaco de Bakker. *Mathematical Theory of Program Correctness*. International Series in Computer Science. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980.

[dB80b] N. G. de Bruijn. A Survey of the Project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, Inc., New York, N.Y., 1980.

[DD85] James Donahue and Alan Demers. Data Types are Values. *ACM Transactions on Programming Languages and Systems*, 7(3):426–445, July 1985.

[Dij75] E. W. Dijkstra. Guarded Commands, Nondeterminancy and Formal Derivation of Programs. *Communications of the ACM*, 18(8):453–457, August 1975.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.

[DM82] L. Damas and R. Milner. Principal Type-Schemes for Functional Programs. In *Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico*, pages 207–212. ACM, January 1982.

[DMN70] Ole-Johan Dahl, B. Myhraug, and K. Nygaard. The Simula 67 common base language. Publication S-22, Norwegian Computing Center, Oslo, Norway, 1970.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and program semantics*. Springer-Verlag, NY, 1990.

[DT88] Scott Danforth and Chris Tomlinson. Type Theories and Object-Oriented Programming. *ACM Computing Surveys*, 20(1):29–72, March 1988.

[Dyb90] Peter Dybjer. Comparing Integrated and External Logics of Functional Programs. *Science of Computer Programming*, 14(1):59–79, June 1990.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, New York, N.Y., 1985.

[End72] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., Orlando, Florida, 1972.

[FB86] Michael Fisher and Howard Barringer. Program Logics – A Short Survey. Technical Report UMCS-86-11-1, Department of Computer Science, University of Manchester, Manchester M13 9PL, England, November 1986. Revised June 1987.

[FF86] Matthias Felleisen and Daniel P. Friedman. Control Operators, the SECD-Machine, and the λ-Calculus. Technical Report 197, Computer Science Department, Indiana University, June 1986.

[FH89] Matthias Felleisen and Robert Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. Technical Report COMP TR89-100, Department of Computer Science, Rice University, December 1989.

[Flo67] R. W. Floyd. Assigning Meanings to Programs. *Proceedings Symposium on Applied Mathematics*, 19:19–31, 1967.

[FWH92] Daniel P. Friedman, Mitchell Wand, and Christopher T. Haynes. *Essentials of Programming Languages*. McGraw-Hill Book Co., New York, N.Y., 1992.

[Ghe91a] Giorgio Ghelli. Modelling Features of Object-Oriented Languages in Second Order Functional Languages with Subtypes. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 311–340. Springer-Verlag, New York, N.Y., 1991.

[Ghe91b] Giorgio Ghelli. A Static Type System for Message Passing. *ACM SIGPLAN Notices*, 26(11):129–145, November 1991. OOPSLA '91 Conference Proceedings, Andreas Paepcke (editor), October 1991, Phoenix, Arizona.

[GHM78] John V. Guttag, Ellis Horowitz, and David R. Musser. Abstract Data Types and Software Validation. *Communications of the ACM*, 21(12):1048–1064, December 1978.

[Gir71] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings 2nd Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland.

[Gir86] J. Y. Girard. The System **F** of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[GL86] David K. Gifford and John M. Lucassen. Integrating Functional and Imperative Programming. In *ACM Conference on LISP and Functional Programming*, pages 28–38. ACM, August 1986.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, New York, N.Y., 1989.

[GM94] Carl A. Gunter and John C. Mitchell, editors. *Theoretical Aspects of Object-Oriented Programming*. Fondations of Computing. The MIT Press, Cambridge, MA, 1994.

[GMP90] David Guaspari, Carla Marceau, and Wolfgang Polak. Formal Verification of Ada Programs. *IEEE Transactions on Software Engineering*, 16(9):1058–1075, September 1990.

[GMW79] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, N.Y., 1979. The second author is listed on the cover as Arthur J. Milner, which is clearly a mistake.

[Gol84] R. Goldblatt. *Topoi: The Categorial Analysis of Logic (Revised Edition)*, volume 98 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, New York, N.Y., 1984.

[Gor79] Michael J. C. Gordon. *The Denotational Description of Programming Languages.* Springer-Verlag, New York, N.Y., 1979.

[Gor88] Michael J. C. Gordon. *Programming Language Theory and its Implementation.* Prentice Hall International Series in Computer Science. Prentice-Hall, Inc., New York, N.Y., 1988.

[GR83] Adele Goldberg and David Robson. *Smalltalk-80, The Language and its Implementation.* Addison-Wesley Publishing Co., Reading, Mass., 1983.

[Gra79] George Gratzer. *Universal Algebra.* Springer-Verlag, New York, N.Y., second edition, 1979.

[Gri81] David Gries. *The Science of Programming.* Springer-Verlag, New York, N.Y., 1981.

[GS90] C. A. Gunter and D. S. Scott. Semantic Domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 8, pages 633–674. North-Holland, New York, N.Y., 1990.

[GS94] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Texts and Monographs in Computer Science. Springer-Verlag, New York, N.Y., 1994.

[Gun92] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* Foundations of Computing. The MIT Press, Cambridge, Mass., 1992.

[Har84] D. M. Harland. *Polymorphic Programming Languages: Design and Implementation.* John Wiley and Sons, New York, N.Y., 1984.

[Har85] Robert Harper. Modules and Persistence in Standard ML. In *Persistence and Data Types: Papers for the Appin Workshop*, pages 419–430. Universities of Glasgow and St. Andrews, Departments of Computer Science, August 1985. Persistent Programming Research Report 16.

[Har86] Robert Harper. Introduction to Standard ML. Technical Report ECS-LFCS-86-14, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, November 1986.

[Hen80] Peter Henderson. *Functional Programming: Application and Implementation.* International Series in Computer Science. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* The MIT Press, Cambridge, Mass., 1988.

[Hen90] Matthew Hennessy. *The Semantics of Programming Languages: an Elementary Introduction using Structural Operational Semantics.* John Wiley and Sons, New York, N.Y., 1990.

[Hes92] Wim H. Hesselink. *Programs, Recursion, and Unbounded Choice*, volume 27 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, New York, N.Y., 1992.

[HHJ$^+$87] C. A. R. Hoare, I. J. Hayes, He Jifeng, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey, and B. A. Sufrin. Laws of Programming. *Communications of the ACM*, 30(8):672–686, August 1987. See corrections in the September 1987 CACM.

[HMM86] Robert Harper, David MacQueen, and Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, March 1986.

[HMT87] Robert Harper, Robin Milner, and Mads Tofte. A Type Discipline for Program Modules. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 250 of *Lecture Notes in Computer Science*, pages 308–319. Springer-Verlag, March 1987.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–583, October 1969.

[Hoa71] C.A.R. Hoare. Proof of a Program: Find. *Communications of the ACM*, 14(1):39–45, January 1971.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1972.

[Hoa89] C. A. R. Hoare. Notes on an Approach to Category Theory for Computer Scientists. In Manfred Broy, editor, *Constructive Methods in Computing Science*, volume F55 of *NATO ASI Series*, pages 245–305. Springer-Verlag, New York, N.Y., 1989.

[How80] W. A. Howard. The Formulae-as-Types notion of Construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., New York, N.Y., 1980.

[HP90] Robert W. Harper and Benjamin C. Pierce. Extensible Records Without Subtypes. Technical Report CMU-CS-90-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, Penn., February 1990. See also the 1991 POPL proceedings.

[Hud89] Paul Hudak. Conception, Evolution, and Application of Functional Programming Languages. *ACM Computing Surveys*, 21(3):359–411, September 1989.

[HW73] C. A. R. Hoare and N. Wirth. An Axiomatic Definition of the Programming Language Pascal. *Acta Informatica*, 2(4):335–355, 1973.

[Jon86] Cliff B. Jones. Program Specification and Verification in VDM. Technical Report UMCS-86-10-5, Department of Computer Science, University of Manchester, Manchester M13 9PL, England, November 1986.

[Jon90] Cliff B. Jones. *Systematic Software Development Using VDM.* International Series in Computer Science. Prentice Hall, Englewood Cliffs, N.J., second edition, 1990.

[Kam90] Samuel N. Kamin. *Programming Languages: An Interpreter-Based Approach.* Addison-Wesley Publishing Co., Reading, Mass., 1990.

[KdRB91] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol.* The MIT Press, Cambridge, Mass., 1991.

[Kni89] Kevin Knight. Unification: A Multidisciplinary Survey. *ACM Computing Surveys*, 21(1):93–124, March 1989.

[Lam89] Leslie Lamport. A Simple Approach to Specifying Concurrent Systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[Lan64] P. J. Landin. The Mechanical Evaluation of Expressions. *Computer Journal*, 6:308–320, 1964. See also Landin's paper "A Lambda-Calculus Approach" in *Advances in Programming and Non-Numerical Computation*, L. Fox (ed.), Pergamon Press, Oxford, 1966.

[Lan65] P. J. Landin. A Correspondence Algol 60 and Church's Lambda Notation. *Communications of the ACM*, 8:89–101, 158–165, 1965.

[Lan66] P. J. Landin. The Next 700 Programming Languages. *Communications of the ACM*, 9(3):157–166, March 1966.

[Lan71] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, N.Y., 1971.

[LG86] Barbara Liskov and John Guttag. *Abstraction and Specification in Program Development*. The MIT Press, Cambridge, Mass., 1986.

[LG88] John M. Lucassen and David K. Gifford. Polymorphic Effect Systems. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, Calif.*, pages 47–57. ACM, January 1988.

[LGH$^+$78] R. L. London, J. V. Guttag, J. J. Horning, B. W. Lampson, J. G. Mitchell, and G. J. Popek. Proof Rules for the Programming Language Euclid. *Acta Informatica*, 10(1):1–26, 1978.

[LS86] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, New York, N.Y., 1986.

[LS87] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification (Second edition)*. John Wiley and Sons, New York, N.Y., 1987.

[LS91] F. W. Lawvere and Stephen H. Schanuel. *Conceptual Mathematics: a first introduction to categories*. Buffalo Workshop Press, Buffalo, NY, 1991.

[LSAS77] Barbara Liskov, Alan Snyder, Russell Atkinson, and Craig Schaffert. Abstraction Mechanisms in CLU. *Communications of the ACM*, 20(8):564–576, August 1977.

[MA86] Ernest G. Manes and Michael A. Arbib. *Algebraic Approaches to Program Semantics*. Springer-Verlag, New York, N.Y., 1986.

[MA89] C. McDonald and L. Allison. Denotational Semantics of a Command Interpreter and their Implementation in Standard ML. *The Computer Journal*, 32(5):422–431, October 1989.

[Mac84] David MacQueen. Modules for Standard ML. In *Proceedings of the Symposium on LISP and Functional Programming, Austin, Texas*, pages 198–207. ACM, August 1984.

[Mac86] David MacQueen. Using Dependent Types to Express Modular Structure. In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida*, pages 277–286. ACM, January 1986.

[McC60] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM*, 3(4):184–195, April 1960.

[McC84] Nancy McCracken. The Typechecking of Programs with Implicit Type Structure. In D. B. MacQueen G. Kahn and G. Plotkin, editors, *Semantics of Data Types: International Symposium, Sophia-Antipolis, France*, volume 173 of *Lecture Notes in Computer Science*, pages 301–315. Springer-Verlag, New York, N.Y., June 1984.

[Mee87] L. G. L. T Meertens, editor. *Program Specification and Transformation.* North-Holland, 1987.

[Mey88] Bertrand Meyer. *Object-oriented Software Construction.* Prentice Hall, New York, N.Y., 1988.

[Mey92] Bertrand Meyer. *Eiffel: The Language.* Object-Oriented Series. Prentice Hall, New York, N.Y., 1992.

[MG90] Carroll Morgan and P. H . B. Gardiner. Data Refinement by Calculation. *Acta Informatica*, 27(6):481–503, May 1990.

[MH88] John C. Mitchell and Robert Harper. The Essence of ML. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, Calif.*, pages 28–46. ACM, January 1988.

[Mil78] Robin Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 94 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, N.Y., 1980.

[Mil84] Robin Milner. A Proposal for Standard ML. In *Conference Record of the ACM Symposium on LISP and Functional Programming, Austin, Texas*, pages 184–197. ACM, August 1984. Also appeared as Tech. Report CSR-157-83, University of Edinburgh, Edinburgh, Scotland, 1983.

[Mil90] Robin Milner. Operational and Algebraic Semantics of Concurrent Processes. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 19, pages 1201–1242. The MIT Press, New York, N.Y., 1990.

[Mit86] John C. Mitchell. Representation Independence and Data Abstraction (preliminary version). In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida*, pages 263–276. ACM, January 1986.

[Mit90] John C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 8, pages 365–458. North-Holland, New York, N.Y., 1990.

[ML75] P. Martin-Löf. An Intuitionistic Theory of Types: Predictive Part. In H. E. Rose and J. C. Sheperdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic*, pages 73–118. North-Holland Publishing Co., New York, N.Y., 1975.

[ML82] Per Martin-Löf. Constructive Mathematics and Computer Programming. In L. J. Cohen et al., editors, *Logic, Methodology, and Philosophy of Science VI (Proceedings of the Sixth International Congress; Hannover, 1979)*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North Holland, Amsterdam, 1982.

[MMM91] John Mitchell, Sigurd Meldal, and Neel Madhav. An extension of Standard ML modules with subtyping and inheritance. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando*, pages 270–278. ACM, January 1991.

[Mor73] James H. Morris, Jr. Protection in Programming Languages. *Communications of the ACM*, 16(1):15–21, January 1973.

[Mor90] Carroll Morgan. *Programming from Specifications.* Prentice Hall International, Hempstead, UK, 1990.

[Mos90] Peter D. Mosses. Denotational Semantics. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 11, pages 577–631. The MIT Press, New York, N.Y., 1990.

[Mos92] Peter D. Mosses. *Action Semantics*, volume 26 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, New York, N.Y., 1992.

[MP85] John C. Mitchell and Gordon D. Plotkin. Abstract Types have Existential Type. In *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages, New Orleans, Louisana*, pages 37–51. ACM, January 1985.

[MPS86] David MacQueen, Gordon Plotkin, and Ravi Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71(1/2):95–130, Oct./Nov. 1986.

[MS76] R. E. Milne and C. Strachey. *A Theory of Programming Language Semantics (part a, part b).* Chapman & Hall, London, 1976.

[MT91] Robin Milner and Mads Tofte. *Commentary on Standard ML.* The MIT Press, Cambridge, Mass., 1991.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* The MIT Press, Cambridge, Mass., 1990.

[MV94] Carroll Morgan and Trevor Vickers, editors. *On the refinement calculus.* Formal approaches of computing and information technology series. Springer-Verlag, New York, N.Y., 1994.

[MW80] Zohar Manna and Richard Waldinger. A Deductive Approach to Program Synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, January 1980.

[MW93] Zohar Manna and Richard Waldinger. *The Deductive Foundations of Computer Programming.* Addison-Wesley, New York, N.Y., 1993.

[NHN80] Reiji Nakajima, Michio Honda, and Hayao Nakahara. Hierarchical Program Specification and Verification — a Many-sorted Logical Approach. *Acta Informatica*, 14(2):135–155, 1980.

[Nip89] T. Nipkow. Formal Verification of Data Type Refinement — Theory and Practice. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands*, volume 430 of *Lecture Notes in Computer Science*, pages 561–591. Spring-Verlag, May/June 1989.

[NN92] F. Nielson and H.R. Neilson. *Semantics with Applications - A Formal Introduction.* John Wiley and Sons, New York, N.Y., 1992.

[NP83] Bengt Nordström and Kent Peterson. Types and Specifications. In R. E. A. Mason, editor, *Information Processing 83*, pages 915–920. Elsevier Science Publishers B.V. (North-Holland), September 1983. Proceedings of the IFIP 9th World Computer Congress, Paris, France.

[OG76] Susan Owicki and David Gries. Verifying Properties of Parallel Programs: An Axiomatic Approach. *Communications of the ACM*, 19(5):279–285, May 1976.

[OG89] James William O'Toole and David K. Gifford. Type Reconstruction with First-Class Polymorphic Values. *ACM SIGPLAN Notices*, 24(7):207–217, July 1989. Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implementation, Portland, Oregon, June.

[OL82] Susan Owicki and Leslie Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.

[PHL$^+$77] G. J. Popek, J. J. Horning, B. W. Lampson, J. G. Mitchell, and R. L. London. Notes on the Design of Euclid. *ACM SIGPLAN Notices*, 12(3):11–18, March 1977. Proceedings of an ACM Conference on Language Design for Reliable Software, Raliegh, North Carolina, March, 1977.

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists.* The MIT Press, Cambridge, Mass, 1991.

[Plo77] G. D. Plotkin. LCF Considered as a Programming Language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo81] Gordon Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Aarhus University, September 1981.

[Rey74] J. C. Reynolds. Towards a Theory of Type Structure. In *Programming Symposium, Proceedings, Colloque sur la Programmation, Paris, April 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, New York, N.Y., 1974.

[Rey75] J. C. Reynolds. User-defined Types and Procedural Data Structures as Complementary Approaches to Type Abstraction. In S. A. Schuman, editor, *New Directions in Algorithmic Languages*, pages 157–168. IRIA, 1975.

[Rey78] John C. Reynolds. User Defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction. In David Gries, editor, *Programming Methodology, A Collection of Articles by IFIP WG2.3*, pages 309–317. Springer-Verlag, New York, N.Y., 1978. Reprinted from S. A. Schuman (ed.), *New Directions in Algorithmic Languages 1975*, Inst. de Recherche d'Informatique et d'Automatique, Rocquencourt, 1975, pages 157-168.

[Rey80] John C. Reynolds. Using Category Theory to Design Implicit Conversions and Generic Operators. In Neil D. Jones, editor, *Semantics-Directed Compiler Generation, Proceedings of a Workshop, Aarhus, Denmark*, volume 94 of *Lecture Notes in Computer Science*, pages 211–258. Springer-Verlag, January 1980.

[Rey85] John C. Reynolds. Three Approaches to Type Structure. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James Thatcher, editors, *Mathematical Foundations of Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Berlin. Volume 1: Colloquium on Trees in Algebra and Programming (CAAP '85)*, volume 185 of *Lecture Notes in Computer Science*, pages 97–138. Springer-Verlag, New York, N.Y., March 1985.

[RL77] Lawrence Robinson and Karl N. Levitt. Proof Techniques for Hierarchically Structured Programs. *Communications of the ACM*, 20(4):271–283, April 1977.

[SCB$^+$86] Craig Schaffert, Topher Cooper, Bruce Bullis, Mike Kilian, and Carrie Wilpolt. An Introduction to Trellis/Owl. *ACM SIGPLAN Notices*, 21(11):9–16, November 1986. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

[Sch82] Oliver Schoett. A Theory of Program Modules, their Specifications and Implementation (Extended Abstract). Internal Report CSR-155-83, Department of Computer Science, University of Edinburgh, December 1982.

[Sch86] David A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, Inc., Boston, Mass., 1986.

[Sch90] Oliver Schoett. Behavioural Correctness of Data Representations. *Science of Computer Programming*, 14(1):43–57, June 1990.

[Sch94] David A. Schmidt. *The Structure of Typed Programming Languages.* Foundations of Computing Series. MIT Press, Cambridge, Mass., 1994.

[Sco81] Dana Scott. Lectures on a Mathematical Theory of Computation. Technical Monograph PRG-19, Oxford University Computing Laboratory, Programming Research Group, 1981. Appears in Theoretical foundations of programming methodology : lecture notes of an international summer school, directed by F.L. Bauer, E.W. Dijkstra, and C.A.R. Hoare (Ridel, 1982).

[Sha81] Mary Shaw. *ALPHARD: Form and Content.* Springer-Verlag, New York, N.Y., 1981.

[SS71] D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In *Proceedings Symposium on Computers and Automata*, volume 21 of *Microwave Institute Symposia Series*, pages 19–46. Polytechnic Institute of Brooklyn, 1971.

[SS78] Guy Lewis Steele Jr. and Gerald Jay Sussman. The Art of the Interpreter or, The Modularity Complex (Parts Zero, One, and Two). AI Memo 453, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1978.

[Sta85] R. Statman. Logical Relations and the Typed λ-Calculus. *Information and Control*, 65(2/3):85–97, May/June 1985.

[Sto77] J. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* The MIT Press, Cambridge, Mass., 1977.

[Str66] C. Strachey. Towards a Formal Semantics. In *IFIP TC2 Working Conference on Formal Language Description Languages for Computer Programming*, pages 198–220, Amsterdam, 1966. North-Holland.

[SWL77] Mary Shaw, William A. Wulf, and R. L. London. Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators. *Communications of the ACM*, 20(8):553–564, August 1977.

[Tal88] C. Talcott. Rum: An intensional theory of function and control abstractions. In M. Boscarol, L. Carlucci Aiello, and G. Levi, editors, *Foundations of Logic and Functional Programming, Workshop Proceedings, Trento, Italy, (Dec. 1986)*, volume 306 of *Lecture Notes in Computer Science*, pages 3–44. Springer-Verlag, 1988.

[Ten76] R. D. Tennent. The Denotational Semantics of Programming Languages. *Communications of the ACM*, 19:437–453, August 1976.

[Tiu90] Jerzy Tiuryn. Type Inference Probelms: A Survey. In B. Rovan, editor, *Mathematical Foundations of Computer Science 1990, Banskà Bystrica, Czechoslovakia*, volume 452 of *Lecture Notes in Computer Science*, pages 105–120. Springer-Verlag, New York, N.Y., 1990.

[Tur85] David A. Turner. Miranda: A non-strict functional language with polymorphic types. In J. Jouannaud, editor, *Proceedings IFIP International Conference on Functional Programming Languages and Computer Architectures, Nancy, France*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, New York, N.Y., September 1985.

[vL90] Jan van Leeuwen. *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. The MIT Press, New York, N.Y., 1990.

[Wal91] R. F. C. Walters. *Categories and Computer Science*, volume 28 of *Cambridge Computer Science Texts*. Cambridge University Press, New York, N.Y., 1991.

[Wan89] Mitchell Wand. Type Inference for Record Concatenation and Multiple Inheritance. In *Fourth Annual Symposium on Logic in Computer Science, Pacific Grove, California*, pages 92–97. IEEE, June 1989.

[Wat86] D. A. Watt. Executable Denotational Semantics. *Software: Practice and Experience*, 16(1):13–43, 1986.

[Wat91] David A. Watt. *Programming Language Syntax and Semantics*. Prentice Hall International Series in Computer Science. Prentice-Hall, New York, N.Y., 1991.

[Weg74] Ben Wegbreit. The Treatment of Data Types in EL1. *Communications of the ACM*, 17(5):251–264, May 1974.

[Win87] Jeannette M. Wing. Writing Larch Interface Language Specifications. *ACM Transactions on Programming Languages and Systems*, 9(1):1–24, January 1987.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages.* Foundations of Computer Science Series. The MIT Press, Cambridge, Mass., 1993.