Introduction to the Literature On Programming Language Design

Gary T. Leavens

TR #93-01c
Jan. 1993, revised Jan. 1994, Feb. 1996, and July 1999

Keywords: programming languages, semantics, type systems, polymorphism, type theory, data abstraction, functional programming, object-oriented programming, logic programming, declarative programming, parallel and distributed programming languages.
1992 CR Categories: D.1.1 [*Programming Techniques*] Applicative (Functional) Programming; D.1.3 [*Programming Techniques*] Concurrent Programming; D.1.5 [*Programming Techniques*] Object-oriented Programming; D.1.6 [*Programming Techniques*] Logic Programming; D.2.2 [*Software Engineering*] Tools and Techniques — Modules and interfaces; D.2.4 [*Software Engineering*] Program Verification; D.2.4 [*Software Engineering*] Program Verification; D.3.1 [*Programming Languages*] Formal Definitions and Theory; D.3.2 [*Programming Languages*] Language Classifications; D.3.3 [*Programming Languages*] Language Constructs and Features; 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; H.2.3 [*Database Management*] Languages—Database (persistent) programming languages; I.2.5 [*Artificial Intelligence*] Programming Languages and Software.

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

Introduction to the Literature On Programming Language Design

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

July 29, 1999

Abstract

This is an introduction to the literature on programming language design and related topics. It is intended to cite the most important work, and to provide a place for students to start a literature search.

This is a selective *introduction* to the literature on programming language design. The intended audience is graduate students beginning a study of programming languages. Instead of trying to be comprehensive, references are given that are to works of lasting value, up-to-date surveys, or that seem to be important or interesting at the moment.[1] Besides references that have intrinsic interest, a few are included because they are the original sources and are likely to be referenced by others doing related work (e.g. (Church 1941)). To probe an area more deeply, start with the papers mentioned, follow their references, and also use the *Science Citation Index* to see what papers have referenced the ones mentioned.

As a general aid to finding papers, many older references are reprinted in a collection called *TUTORIAL Programming Language Design* edited by A. I. Wasserman and available from the IEEE Computer Society (Wasserman 1980). Readers with web browsers may also want to look at the following web page, titled "Programming Language and Compiler Bibliographies."

`http://www.cs.cmu.edu/~mleone/language/bibliographies.html`

However, this tends to have better coverage of recent technical reports, and its coverage of older journal papers and other published material is spotty.

1 Generalities

There are several excellent undergraduate texts on programming languages, including: (Pratt and Zelkowitz 1996) (Sebesta 1996) (Sethi 1996) (Friedman, Wand, and Haynes 1992) (Hennessy 1990) (Kamin 1990) (Watt 1990) (Ghezzi and Jazayeri 1987) (MacLennan 1999) (Tennent 1981). Graduate texts include: (Slonneger and Kurtz 1995) (Stanisfer 1995) (Schmidt 1994) (Winskel 1993) (Watt 1991) (Meyer 1990) (Gordon 1988).

Volume B of the *Handbook of Theoretical Computer Science* contains many detailed surveys relevant to formal models and semantics (van Leeuwen 1990).

Journals include a substantial coverage of programming languages include *ACM Transactions on Programming Languages and Systems* (abbreviated TOPLAS), *ACM Letters on Programming Languages and Systems* (abbreviated LOPLAS), *ACM SIGPLAN Notices*, *Computer Languages* (Pergamon Press), *Journal of Programming Languages* (Chapman and Hall), *Information and Computation* (formerly *Information and*

*This work was supported in part by the National Science Foundation under Grants CCR-9108654, CCR-9593168, and CCR-9503168.

[1]Naturally these judgements are personal, but it is hoped that authors of that vast bulk of papers that are not cited here will not take offense.

Control Academic Press), and *Acta Informatica* (Springer-Verlag). Applied areas, such as specification and verification have their own journals (e.g., *IEEE Transactions on Software Engineering*, *Science of Computer Programming*, *Formal Aspects of Computing*). Other journals can be found by scanning the bibliography.

Conferences devoted to topics related to programming languages include the Annual ACM Symposium on Principles of Programming Languages (POPL), the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), the ACM SIGPLAN International Conference on Functional Programming (formerly called the Symposium on LISP and Functional Programming, LFP), and the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP). Conferences with a more theoretical bent include the IEEE Annual Symposium on Logic in Computer Science (LICS), Mathematical Foundations of Programming Semantics (MFPS), the European Symposium on Programming (ESOP), 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). The above list does not include conferences devoted to software engineering (e.g. TAPSOFT) or particular methods (e.g., category theory) where other important work is found.

Both Hoare and Wirth have written guidelines for programming language designers (Hoare 1980) (Wirth 1974). Floyd's Turing Award lecture discusses his view of programming languages as capturing paradigms of programming (Floyd 1979). Another viewpoint is that language design should proceed from semantic principles (Tennent 1977) (Backus 1978a).

The truism that bad programs can be written in any programming language is described as "Flon's Axiom" in a paper written at the height of the structured programming controversy (Flon 1975). Shaw and Wulf point out that language designers can provide reasonable defaults while still giving programmers the ability to change them (Shaw and Wulf 1980).

The idea of the expressiveness of a language is just starting to receive satisfactory formal treatments (Felleisen 1990) (Mitchell 1991).

2 Semantics

Some texts that cover many different approaches to semantics are: (Nielson and Neilson 1992) (Winskel 1993) (Slonneger and Kurtz 1995).

2.1 Operational Semantics

Landin's SECD machine is defined in his paper "The Mechanical Evaluation of Expressions" (Landin 1964) (see also Henderson's book (Henderson 1980)).

A more systematic style of operational semantics based on rewrite rules is found in Plotkin's structural operational semantics (Plotkin 1977) (Hennessy 1990) (Astesiano 1991).

Several meta-circular interpreters for variants of LISP are discussed in Steele and Sussman's paper *The Art of the Interpreter* (Steele Jr. and Sussman 1978a). An excellent and more readily accessible discussion is found in Abelson and Sussman's book (Abelson, Sussman, and Sussman 1996), which uses Scheme. A more detailed treatment of interpreters is found in (Kamin 1990) (Friedman, Wand, and Haynes 1992). See (Kiczales, des Rivieres, and Bobrow 1991) for an approach using the CLOS meta-object protocol. More recently, structuring mechanisms for semantics based on monads (Moggi 1990) (Wadler 1992) (Wadler 1997) have lead to more modular structuring of definitional interpreters (Steele, Jr. 1994) (Liang, Hudak, and Jones 1995).

2.2 λ-Calculus

A standard reference on Church's λ-calculus (Church 1941) is Barendregt's book (Barendregt 1984). Informal introductions include (Gordon 1988). For the typed λ-calculus, see (Girard, Lafont, and Taylor 1989) or (Mitchell 1990b).

The use of λ-calculus for describing programming languages and as the inspiration for programming language design has been investigated by Landin (Landin 1965) (Landin 1966).

2.3 Denotational Semantics

A short summary of the denotational approach to programming language semantics (Scott 1981) can be found in Tennent's article "The Denotational Semantics of Programming Languages" (Tennent 1976). A survey is found in (Mosses 1990).

Introductory texts include (Gordon 1979), (Allison 1986), (Watt 1991) (Schmidt 1994). An excellent text with mathematical depth is (Gunter 1992). Standard works on denotational semantics are the books by Stoy (Stoy 1977) and Schmidt (Schmidt 1986), both of which offer a comprehensive and mathematical treatment. Schmidt's book (Schmidt 1986) can be consulted for references to denotational descriptions of real languages.

To read some of the most technical works, one will need some familiarity with category theory (Lane 1971) (Goldblatt 1984) (Lawvere and Schanuel 1991); the following are good introductions that emphasize semantic applications (Hoare 1989) (Barr and Wells 1990) (Asperti and Longo 1991) (Pierce 1991) (Walters 1991) (Gunter 1992).

One can use a typed functional programming language, such as Standard ML, to implement a denotational semantics. Two descriptions of this are (Watt 1986) (McDonald and Allison 1989).

Action semantics, an offshoot of denotational semantics, is described in (Watt 1991) and more fully in (Mosses 1992).

2.4 Axiomatic Semantics

An early presentation of axiomatic semantics is Hoare's paper "An Axiomatic Basis for Computer Programming" (Hoare 1969).

An example of the use of axiomatic semantics to define a programming language is Hoare and Wirth's axiomatic definition of Pascal (Hoare and Wirth 1973). A book on axiomatic semantics slanted towards programming language theory (as opposed to verification) is Hesselink's monograph (Hesselink 1992). Foundational material that may help in reading this monograph is found in (Dijkstra and Scholten 1990), which goes over notational issues as well as the underlying mathematics.

A very introductory tutorial on verification from the software engineering perspective is (Liskov and Guttag 1986, 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 (Dijkstra 1976) (Gries 1981). More recent treatments in this style advocate a calculational approach (Morgan 1994) (Cohen 1990) (Gries and Schneider 1994). See (Apt and 0lderog 1991) for an introduction that discusses concurrent and distributed programs.

A survey of program verification for imperative programs is (Cousot 1990).

A language designed to support program verification is Euclid (London *et al.* 1978) (Popek *et al.* 1977). Another such language is Alphard (Shaw, Wulf, and London 1977) (Shaw 1981).

2.5 Algebraic Semantics

Another approach to the semantics of programming languages is the algebraic approach; specifying the behavior of programs instead of directly specifying the function computed. A book-length treatment is (Bergstra, Heering, and Klint 1989), which uses tools described in (Klint 1993). Examples include (Manes and Arbib 1986) (Broy, Wirsing, and Pepper 1987) (Prasad, Giacalone, and Mishra 1990) (Slonneger and Kurtz 1995, Chapter 12).

3 Type Systems

3.1 Background

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 (Morris 1973). Wegbreit's discussion of the extensible language EL1, is also good background (Wegbreit 1974).

3.2 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" (Cardelli and Wegner 1985). See also (Harland 1984) (Cardelli 1991) (Danforth and Tomlinson 1988); the latter two have much material related to object-oriented programming. A still more recent survey is (Mitchell 1990b).

Standard references include Girard's system Fω (Girard 1971) (see also (Girard 1986) (Girard, Lafont, and Taylor 1989)), and Reynolds independent work (Reynolds 1974), sometimes called the Girard-Reynolds second order lambda calculus (or SOL). Modern expositions are found in (Mitchell and Plotkin 1985) (Mitchell and Harper 1988) (Reynolds 1985) (Mitchell 1990b) (Cardelli 1991) (Schmidt 1994).

Other kinds of type information may be incorporated into a type system and checked at the same time as types (Gifford and Lucassen 1986) (Lucassen and Gifford 1988) (O'Toole and Gifford 1989).

3.3 Type Inference

Type inference is also sometimes called type reconstruction The basic idea is described in Milner's paper "A Theory of Type Polymorphism in Programming" (Milner 1978). A more modern exposition of Milner's ideas is (Cardelli 1987). A top-down variation of the standard algorithm is studied in (Lee and Yi 1998).

This type inference system is used in the programming language Standard ML (Gordon, Milner, and Wadsworth 1979) (Milner, Tofte, and Harper 1990) (Milner and Tofte 1991). Textbooks about programming in Standard ML include (Paulson 1991) (Sokolowski 1991) (Stanisfer 1992) (Ullman 1994). ML also has a very interesting module system (Harper and Lillibridge 1994) (Leroy 1994) (MacQueen 1984).

An important research problem (Hoang and Mitchell 1995) has been extending this type system to handle subtyping (Eifrig, Smith, and Trifonov 1995) (Nordlander 1999) (Odersky, Sulzmann, and Wehr 1999) (Pierce and Turner 1998) (Pottier 1999).

An extension of this type system to handle static overloading (*ad hoc* polymorphism) is described in(Wadler and Blott 1989) and further refined in (Jones 1995) (Nipkov and Prehofer 1994). One way to handle mutation in this type system is described in (Harper 1994). There are also object-oriented extensions to ML (Rémy and Vouillon 1998) (Reppy and Riecke 1996) (Fischer and Reppy 1999).

The language Poly (Matthews 1985b) (Matthews 1985a) uses type inference to infer type parameters. An algorithm for inferring operation parameters is described in (Cormack and Wright 1990). Other extensions to the basic type inference algorithm are described in (O'Toole and Gifford 1989) (Talpin and Jouvelot 1994) (Boehm 1989). Along similar lines, one can use a mix of type annotations and inference to allow polymorphic functions to be passed as arguments (Odersky and Läufer 1996).

The computational complexity of various type checking and type inference problems has also been an active area of research (Wells 1994) (Schubert 1998) (Henglein 1999). See (Tiuryn 1990) for a survey.

3.4 Type Theory

Type theory, narrowly defined, uses the tools of constructive logic to study type systems such as the above. Logical inference systems can often be translated directly into type systems due to the "Formula as Types" notion or the "Curry-Howard isomorphism" (Howard 1980) (Girard, Lafont, and Taylor 1989, Chapter 3) (Constable 1989). 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 (Nordström and Peterson 1983) (Dybjer 1990).

After reading the references above, one will still need an introduction to some of the more technical aspects of type theory. Good book-length treatments are (Thompson 1991) and (Schmidt 1994). Those wishing a shorter introduction might try (Reynolds 1985) (which is not comprehensive, but is tutorial) (Backhouse 1989) (which especially focuses on Martin-Löf style type theory) and (Scedrov 1990) (which focuses more on the calculus of constructions). After reading one of these the student may want to read (Pierce, Dietzen, and Michaylov 1989) for some practical hints.

In the past, some groups working on type theory have included deBruijn and others working on AUTOMATH (de Bruijn 1980), Martin-Löf's and followers (Martin-Löf 1975) (Martin-Löf 1982) (Backhouse 1989)

(Backhouse *et al.* 1989), Constable's group at Cornell (Constable and Zlatin 1984), and Coquand and Huet's group (Coquand and Huet 1988) has also been influential for modern language design.

Some of the latest work on type theory uses linear logic (Girard 1993) instead of a more standard logic. A linear type system allows a value to be used only once (Baker 1991) (Mackie 1993) (Smetsers *et al.* 1994) (Kobayashi 1999).

There are also connections between type theory and abstract interpretation (Cousot 1997) (Palsberg and Pavlopoulou 1998).

3.5 Data Abstraction and Types

A good introductory treatment of the idea of data abstraction is found in (Liskov and Guttag 1986). A more technically oriented introduction is (Cardelli and Leroy 1990). Another excellent paper is (Cook 1991), which distinguishes between programming with abstract data types and object-oriented programming.

CLU is a language designed around data abstraction (Liskov *et al.* 1977) (Liskov *et al.* 1981) (Liskov and Guttag 1986). It also has an interesting control abstraction and exception handling mechanisms (Liskov and Snyder 1979).

Alphard, designed around the same time as CLU and with many of the same goals, has surprising differences (Shaw, Wulf, and London 1977) (Shaw 1981).

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" (Donahue and Demers 1985), which can be consulted for other references.

Much recent work involves object-oriented languages. For example, Trellis/Owl (Schaffert *et al.* 1986) features strong type checking and a declared (i.e., by-name) subtype relation. By contrast many other languages feature structural subtyping, including Emerald (Black *et al.* 1986) (Black *et al.* 1987) (Black and Hutchinson 1990) (Black and Hutchinson 1991) (Cardelli 1991). Two excellent books, one by Abadi and Cardelli (Abadi and Cardelli 1996) and another by Castagna (Castagna 1997), are a good starting point in this area. Other theoretical work in this area includes the following (Bruce, Cardelli, and Pierce 1996) (Cardelli 1988b) (Cardelli 1988a) (Amadio and Cardelli 1993) (Cardelli *et al.* 1994) (Canning *et al.* 1989) (Cook, Hill, and Canning 1990) (Cook 1989) (Breazu-Tannen, Gunter, and Scedrov 1990) (Bruce *et al.* 1993) (Bruce and Mitchell 1992) (Bruce 1993) (Bruce *et al.* 1993) (Palsberg and Schwartzbach 1994) (Bruce 1994) (Pierce and Turner 1994) (Abadi 1994) (Abadi and Cardelli 1994) (Abadi and Cardelli 1995) (Eifrig, Smith, and Trifonov 1995) (Fischer and Mitchell 1998). (Cardelli has been 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 (Reynolds 1980) (Ghelli 1991a) (Ghelli 1991b) (Castagna, Ghelli, and Longo 1992) (Chambers 1992) (Castagna, Ghelli, and Longo 1995) (Castagna 1993) (Castagna 1995) (Castagna 1997) (Chambers and Leavens 1995) (Leavens and Pigozzi 1999) (Millstein and Chambers 1999). For adding multimethods to conventional languages see (Boyland and Castagna 1997) (Leavens and Millstein 1998). For a tutorial discussion of the problems of typing binary methods, see (Bruce *et al.* 1995).

Programming languages with separate compilation do some of their type checking at link-time (Liskov *et al.* 1981) (Leroy 1994) (Stroustrup 1991). This interaction of types, separate compilation, linking, and modules has recently been formalized (Cardelli 1997) (Glew and Morrisett 1999). Some recent theoretical work been on combining modules and object-oriented programming features (Findler and Flatt 1999) (Fischer and Reppy 1999) (Millstein and Chambers 1999).

4 Alternative Programming Models

4.1 Functional Programming

A survey of functional programming is (Hudak 1989), which also discusses the language Haskell (Hudak and Fasel 1992) (Hudak *et al.* 1992) (Snow 1992) (Davie 1992). Another survey is (Barendregt 1990). There are several good books on functional programming, including (Henson 1987) (Bird and Wadler 1988) (Henderson 1980) (Springer and Friedman 1989) (Okasaki 1998). For an introduction that also treats language implementation issues, see (Peyton Jones 1987). The articles in (Turner 1990b) make an interesting introduction to some research topics.

John Backus, one of the designers of Fortran, proposed a new language for functional programming without any names called FP in his Turing Award Lecture (Backus 1978a). A functional programming style using a more congenial notation based on Landin's ISWIM (Landin 1966) is developed in Henderson's book *Functional Programming: Application and Implementation* (Henderson 1980).

One axis of variation in functional languages is between the lazy and strict (eager) languages (Wadler 1996). ML is eager, but Haskell and its predecessor Miranda (Turner 1990a) are lazy. Miranda also has an interesting notion of data abstraction.

Recently, various approaches to incorporating state information in a safe way into functional languages have centered around the use of monads (Launchbury and Sabry 1997) (Moggi 1990) (Wadler 1992) (Wadler 1997). There is syntactic support for monads in Haskell. Others are exploring alternatives to monads (Chen and Hudak 1997) (Kagawa 1997) (Odersky 1999) (Smetsers *et al.* 1994) (Wadler 1999).

Erlang (Armstrong *et al.* 1995) is used by Ericsson telephone company in several commercial products.

4.2 Logic Programming

Kowalski's paper "Algorithm = Logic + Control" is a good introduction to logic programming in an idealized setting (Kowalski 1979). A classic textbook on Prolog is the book by Clocksin and Mellish (Clocksin and Mellish 1981); another good text is Sterling and Shapiro's (Sterling and Shapiro 1994). A short description and evaluation of Prolog is found in the paper "The Prolog Phenomenon" (McDermott 1980). A survey is (Apt 1990).

Several "AI languages" preceded the development of Prolog; for example, Planner (Sussman, Winograd, and Charniak 1971) and Conniver (McDermott and Sussman 1974).

Much work has focused on concurrent logic programming languages (Shapiro 1989), which are perhaps more like CSP than logic programming. An evaluation of the Fifth-Generation project and some history of concurrent logic programming languages is found in (Shapiro and Warren 1993). A related language is Andorra Prolog (Brand, Haridi, and Warren 1989).

Work on type checking for logic programming languages is surveyed in the collection (Pfenning 1992). The language λProlog features type inference, type checking, and higher-order programming constructs (Miller 1990) (Miller 1989) (Nadathur and Miller 1990) (Miller *et al.* 1991).

4.3 Other Declarative Programming Paradigms

Constraint-based languages will probably be important in the future. An early attempt was embodied in Steele and Sussman's work, as described in (Sussman and Steele 1980). An overview is given in Leler's book (Leler 1988). A survey is found in (van Hentenryck, Saraswat, et al. 1996). The language CLP($\mathcal{R}$) is a well-known constraint logic programming language (Jaffar *et al.* 1992).

Some higher-order equational logic languages based on variations of narrowing (Slagle 1974) have started to appear in research languages (Miller 1991) (Pfenning 1991) (Qian 1994).

Languages based on term rewriting without logic variables also allow for parallelism. A standard example is OBJ (Futatsugi *et al.* 1985), which also has an interesting module system (Goguen 1984).

4.4 Object-Oriented Programming

A good, but not very technical, introduction to object-oriented concepts is given by Cox (Cox 1986); his book also discusses the language Objective-C. A more technical introduction is Budd's book (Budd 1991b). Meyer's book on Eiffel also has more technical meat (Meyer 1988), as well as a focus on software engineering concerns. Another fairly complete treatment is given in Goldberg and Robson's book on Smalltalk-80 (Goldberg and Robson 1983). A graduate-level introduction is (Blair *et al.* 1991).

Descriptions of object-oriented design methods are found in (Booch 1991) (Wirfs-Brock, Wilkerson, and Wiener 1990) (de Champeaux, Lea, and Faure 1992) (de Champeaux and Faure 1992) (de Champeaux, Lea, and Faure 1993). Both (Booch 1991) and (de Champeaux, Lea, and Faure 1993) have many references. A treatment of object-oriented design that focuses more on C++ is found in (Mullin 1989). See also the survey by (Wirfs-Brock and Johnson 1990). Much recent work in this area has focused on design patterns (Johnson 1992) (Gamma *et al.* 1995).

A collection of papers is found in (Peterson 1987). More collections of edited research papers are (Shriver and Wegner 1987) (Kim and Lochovsky 1989).

The major annual conferences on object-oriented programming are the *European Conference on Object-Oriented Programming* (ECOOP) and *Object-Oriented Programming Systems, Languages and Applications* (OOPSLA). The ECOOP tends to be more academic, while OOPSLA is more practical. The OOPSLA proceedings have been published as special issues of ACM *SIGPLAN Notices* (November 1986, December 1987 with an addendum in May 1988, and November 1988, and October of the following years). ECOOP and OOPSLA had a joint conference in 1990.

There are now three journals devoted to object-oriented programming. The *Journal of Object-Oriented Programming* (JOOP) is the oldest. Two more academic journals are *Theory and Practice of Object Systems* (TAPOS), and *Object-Oriented Systems*.

The best documented and the cleanest object-oriented programming language is Smalltalk-80 (Ingalls 1981) (Goldberg and Robson 1983). Squeak is a recent dialect (Ingalls *et al.* 1997).

A more widely used language is C++ (Ellis and Stroustrup 1990). Good introductions to C++ include (Stroustrup 1997) and (Lippman 1991).

Multiple inheritance, a feature not found in Smalltalk-80, seems to be quite useful. Snyder's analysis of the design issues involved is insightful (Snyder 1986), although his viewpoint is different from that of most advocates of multiple inheritance.

Semantics of inheritance (as opposed to type theory or semantics of subtyping) appear in (Kamin 1988) (Bracha and Cook 1990) (Cook and Palsberg 1989). See (Taivalsaari 1996) for a survey on the notion of inheritance.

My own views on the subject of object-oriented specification, verification, and subtyping can be found in (Leavens and Weihl 1990) (Leavens 1991b) (Leavens and Weihl 1995). For a contrast, see also (Bruce and Wegner 1990) (America 1987) (America and van der Linden 1990) (Meyer 1988) (Marti-Oliet and Meseguer 1990) (Cook 1992) (Utting and Robinson 1992) (Liskov and Wing 1994).

The concept of delegation is explored in (Lieberman 1986), in Actra (LaLonde, Thomas, and Pugh 1986) (LaLonde 1989), and in Ungar's language "Self" (Ungar and Smith 1987).

The contrast between message passing and other kinds of polymorphism is one of binding time. Some relevant semantic models are discussed in (Mitchell 1990a) and (Cook, Hill, and Canning 1990).

A dated survey of the literature on object-oriented programming is (Leavens 1991a).

4.5 Blends of Various Paradigms

Various authors have tried to blend various paradigms. Blends of functional and logic programming are found in (Goguen and Meseguer 1986) (Jayaraman and Gupta 1989). Blends of imperative and logic programming ideas are found in (Budd 1991a) (Apt and Schaerf 1998). Goguen and Meseguer even try to unify everything (Goguen and Meseguer 1987).

Another approach is a multiparadigm language. Leda is one example (Budd 1995).

5 Language Case Studies

The following entries are intended to be selective rather than comprehensive. Instead they are biased towards the most interesting languages and references for the programming language designer. Thus, although COBOL (ANSI 1974) was (is?) the most widely used language on the planet, its influence on programming language design has been small. While the languages discussed below are often obscure, they demonstrate interesting issues in language design.

Those interested in history for its own sake, or in delving further into early languages, should look at the proceedings of the two History of Programming Languages Conferences (Wexelblat 1978) (Wexelblat 1993). The first conference covers the earliest languages, including COBOL, BASIC, and many others not discussed below. Many of the more established languages have their definitions standardized. These are often published by the American National Standards Institute (ANSI), the International Standards Organization (ISO), or the IEEE.

Other resources for case studies include Kamin's book (Kamin 1990), which has several case studies put in a common framework. The "Grand Tour" book by Horowitz has articles about specific languages as well (Horowitz 1987). See also the languages mentioned under the various paradigms above.

5.1 FORTRAN

The first widely used programming language was FORTRAN. See (Backus 1978b) for a discussion of the history of FORTRAN, what early versions of FORTRAN were like, and early references. The development of FORTRAN IV is discussed in (Backus and Heising 1964).

John Backus, who headed the team that developed FORTRAN, later became dissatisfied with the influence that FORTRAN had on programming languages (Backus 1978a).

5.2 Algol 60

The Algol 60 report is a true classic (Naur *et al.* 1963). Among other innovations, it introduced the syntax formalism now known as BNF. Despite the precise use of English in the report, Knuth and others were able to find problems with the language definition (Knuth 1967).

5.3 Algol 68

Algol 68 is a direct descendent of Algol. It is a more powerful and more complete language than Algol 60; for example, it has user-defined types, overloading of operators, and mechanisms for parallel processing. The language design is fascinating and bristles with examples of orthogonality (one of many terms coined in the Algol 68 design). The revised report is a forbidding document, which has an innovative formal mechanism for defining the language's semantics (van Wijngaarden *et al.* 1977). Because the revised report is difficult to follow, Tanenbaum's tutorial is probably a better place to start (Tanenbaum 1976). Those seriously interested in Algol 68 will want to consult (Lindsey and van der Meulen 1977).

5.4 C

A popular descendent of Algol 68 is the lower-level language C (Kernighan and Ritchie 1978). C represents the best of several languages that support low-level programming while maintaining the portability of the resulting program. The language has recently been standardized, and incorporates several changes from its variant C++ (Stroustrup 1991).

5.5 Algol W, Pascal, Modula-2, and Oberon

Wirth and Hoare's language known as Algol W can be thought of as an improved version of Algol 60 (Wirth and Hoare 1966). Wirth's language Pascal (Wirth 1971) (Jensen and Wirth 1974) has been enormously popular, attracting detractors (Habermann 1973) and defenders (Lecarme and Desjardins 1975). Pascal is, in part, a response to the complexity of Algol 68. All this attention has provoked Wirth to reassessing Pascal (Wirth 1975) and to the design of Modula-2 (Wirth 1985), and Oberon (Wirth 1988). Oberon has object-oriented features, as does the (non-Wirth) language Modula-3 (Cardelli *et al.* 1989) (Nelson 1991).

5.6 Euclid

Euclid is an attempt to improve on Pascal in a different direction (Popek *et al.* 1977) (London *et al.* 1978). Specifically, it attempts to support program verification.

5.7 Ada

Ada was designed by first setting out requirements for the language (High Order Language Working Group, Department of Defense 1978) and then designing and revising a language to meet those requirements (Ichbiah *et al.* 1979) (Ada 1983) (Ichbiah *et al.* 1991). Recently, the language was revised to add some object-oriented features (Barnes, Brosgol, et al. 1995) (Ada 1995).

5.8 Java

Java (Gosling, Joy, and Steele 1996) (Arnold and Gosling 1998) is an object-oriented language with a syntax similar to C++. However underneath, it is like Lisp, as it features garbage collection and implicit pointers. It also has reflective features, including dynamic class loaders (Liang and Bracha 1998). Although it has a strong, static, and safe (Drossopoulou, Eisenbach, and Khurshid 1999) (Nipkow and von Oheimb 1998) type system, because it has a universal type (`Object`) that is a supertype of (almost) all the other types and an operation to dynamically check types (`instanceof`, with checked type casts), it can be programmed as if it were a dynamically-typed language. Several authors have proposed extensions to add parametric polymorphism to the language (Bracha *et al.* 1998) (Cartwright and Steele Jr. 1998) (Myers, Bank, and Liskov 1997) (Odersky and Wadler 1997) (Thorup 1997).

5.9 Lisp-like Languages

The original LISP is described in the *LISP 1.5 Programmer's Manual* (McCarthy *et al.* 1965). The language has since evolved in many directions. MacLisp is a main-stream dialect that provided many system building tools (Pitman 1983). The successor to MacLisp is Common Lisp (Steele Jr. 1984) (Steele Jr. 1990). Unlike most earlier dialects of Lisp, Common Lisp has static scoping.

Scheme was the first dialect of Lisp to emphasize static scoping (Steele Jr. and Sussman 1978b) (Abelson, Sussman, and Sussman 1996). See also the references in the *Revised^5 report* (Kelsey, Clinger, and (Editors) 1998).

ZetaLisp (as found on Symbolics Lisp Machines) includes the influential *Flavors* mechanism for object-oriented programming (Weinreb and Moon 1980) (Stallman, Moon, and Weinreb 1984) (Symbolics, Inc. 1984). The Flavors mechanism evolved into the Common Lisp Object System (Keene 1989) (Steele Jr. 1990) (Paepcke 1993). The meta-object system of Common Lisp is described in (Kiczales, des Rivieres, and Bobrow 1991); it provides a very flexible way to extend the language. The ideas in the meta-object system have led to Aspect-Oriented Programming (Kiczales *et al.* 1997) that is designed to allow the isolation of concerns that would otherwise be spread throughout a program.

5.10 Snobol, SL5, and Icon

SNOBOL4 is a language for string processing that is unlike any other (Griswold, Poage, and Polonsky 1971). Because it is so unstructured, its designers have tried to place its powerful features in a more structured framework. The language SL5, one attempt in this direction, is notable for its flexible procedure mechanism (Hanson and Griswold 1978). Another descendent of SNOBOL4, Icon, is a more conventional programming language with innovative control structures (Griswold and Griswold 1983).

5.11 APL

Another unconventional language is APL (Iverson 1962) (Iverson 1991). APL has array processing features without equal and generic operators that can be combined in interesting ways (Iverson 1979). More references on APL can be found by consulting the proceedings of the yearly *International Conference on APL* (sponsored by the ACM). (In the past this was often just called *APL 83* or some such name.)

The most recent version of APL, APL2, is described in (Brown 1988) (Brown and Crowder 1991).

6 Parallel and Distributed Programming Languages

Parallel programming languages are a hot topic of current research, and one with considerable overlap with operating systems, networking, and database systems. Many of the above areas and articles have implications for parallel programming. This area also has several journals and conferences of its own. Some journals that publish programming language related materials but which were not mentioned above include *Distributed Computing*, *IEEE Transactions on Parallel and Distributed Systems*, and *International Journal of Parallel Programming*. See also *ACM SIGOPS Operating Systems Review*. Important conferences not mentioned above include the annual *ACM Symposium on Operating Systems Principles*, *ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, and the *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*.

A recent survey is (Skillicorn and Talia 1998). An older, but still good, survey on parallel programming language issues is (Andrews and Schneider 1983). A survey that focuses on the Linda model appears in (Carriero and Gelernter 1989). (See (Butcher and Zedan 1991) (Butcher and Zedan 1992) (Gelernter and Carriero 1992) for more work on Linda.) An older survey that focuses on object-oriented aspects is (Wyatt, Kavi, and Hufnagel 1992), and a more recent survey (Briot, Guerraoui, and Lohr 1998) also discusses distributed object-oriented programs. A survey that focuses on distributed programs is (Andrews 1991b). Andrew's textbook on concurrent programming languages also contains material on program verification as well as some survey material (Andrews 1991a). Data flow languages are surveyed in (Ackerman 1982). A useful collection of papers is (Skillicorn and Talia 1995). An introduction to concurrent programming techniques is (Snow 1992).

Actor languages are discussed in (Agha 1991). A concurrent version of ML is described in (Reppey 1993). Some aspects of data parallel programming languages are described in (Hillis and Steele Jr. 1986) (Grimshaw 1993). The language Jade is described in (Rinard and Lam 1992) (Rinard, Scales, and Lam 1993).

Data-parallel languages allow one to split data among processors, and to execute the same code for each. A prominent example is High Performance Fortran (Loveman 1993). The language Orca combines both data and task parallelism (Hassan, Bal, and Jacobs 1998).

Distributed programming is a subarea of parallel programming with its own set of problems (Mullender 1993). A survey appears in (Bal, Steiner, and Tanenbaum 1989), which can be consulted for other references. A survey, with a focus on object-orientation is (Chin and Chanson 1991). Other sources of references are is the collection of reprints (Ananda and Srinivasan 1991), and the book (Gray and Reuter 1993). Of particular interest are the languages Argus (Liskov and Scheifler 1983) (Weihl 1990) and SR (Andrews *et al.* 1988) (Andrews and Olsson 1993), The Amoeba distributed operating system also has interesting implications for language designers (Tanenbaum *et al.* 1990) (Tanenbaum, Kaashoek, and Bal 1992).

Mobile code is a recent research interest. It was featured in Emerald (Black *et al.* 1986) (Black *et al.* 1987) (Black and Hutchinson 1990) (Black and Hutchinson 1991), and of course is part of Java. Some other mobile languages of interest include Obliq (Cardelli 1995), Mobile UNITY (Roman 1998), and Distributed Oz (van Roy *et al.* 1997). A recent survey of the field is (Thorn 1997).

An older but still heavily used approach to the semantics of concurrent processes is Petri nets (Peterson 1977) (Papelis and Casavant 1992). A classic reference for the operational semantics of concurrent processes is Milner's book on CCS (Milner 1989). Another widely used approach is Hoare's CSP (Hoare 1978) (Brookes, Hoare, and Roscoe 1984) (Lamport and Schneider 1984) (Hoare 1985). See (Hennessy 1988) (Bloom and Meyer 1990) (Milner 1990) (Milner, Parrow, and Walker 1992) for more work in the semantics of concurrency, and (Lamport and Lynch 1990) and (Broy 1991) for work in the semantics of distributed systems.

7 The Future

A (by now a bit dated) summary of research directions for language design is given in (Lindstrom, Liskov, and MacQueen 1989).

Consumers of programming languages (programmers and language standardization committees) seem to be fairly conservative, and interested more in performance than elegance or expressive power. See (Gabriel 1993) for a pessimistic view of what this means for language design.

Acknowledgements

Thanks to my teachers. Thanks especially to Lawrence Flon, who introduced me to the literature on programming languages. Thanks also to Barbara Liskov, David Gifford, and William Weihl, who helped shape my attitudes towards it. Thanks to students of the class Com S 541 at Iowa State University who have given me feedback on the literature recommendations contained herein.

References

Abadi, Martín and Luca Cardelli. 1994 (April). A theory of primitive objects — untyped and first-order systems. In Hagiya, Masami and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, Vol. 789 of *Lecture Notes in Computer Science*, pages 296–320. Springer-Verlag.

———. 1995. On subtyping and matching. In Olthoff, Walter, editor, *ECOOP '95: Object-Oriented Programming 9th European Conference, Aarhus, Denmark*, number 952 in Lecture Notes in Computer Science, pages 145–167. New York, NY: Springer-Verlag.

———. 1996. *A Theory of Objects*. Monographs in Computer Science. New York, NY: Springer-Verlag.

Abadi, Martín. 1994 (April). Baby modula-3 and a theory of objects. *Journal of Functional Programming*, 4(2):249–283.

Abelson, Harold, Gerald Jay Sussman, and Julie Sussman. 1996. *Structure and Interpretation of Computer Programs*. second edition. Cambridge, Mass.: McGraw Hill.

Ackerman, W. B. 1982 (February). Data flow languages. *Computer*, 15(2):15–25.

American National Standards Institute. 1983 (February). *Reference Manual for the Ada Programming Language*. ANSI/MIL-STD 1815A. Also published by Springer-Verlag as LNCS 155.

International Organization for Standardization. 1995 (January). *Ada 95 Reference Manual. The Language. The Standard Libraries.* ANSI/ISO/IEC-8652:1995.

Agha, Gul. 1991. The structure and semantics of actor languages. In de Bakker, J. W., W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, Vol. 489 of *Lecture Notes in Computer Science*, pages 1–59. New York, NY: Springer-Verlag.

Allison, Lloyd. 1986. *A Practical Introduction to Denotational Semantics*. New York, NY: Cambridge University Press.

Amadio, Roberto M. and Luca Cardelli. 1993 (September). Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631.

America, Pierre and Frank van der Linden. 1990 (October). A parallel object-oriented language with inheritance and subtyping. *ACM SIGPLAN Notices*, 25(10):161–168. *OOPSLA ECOOP '90 Proceedings*, N. Meyrowitz (editor).

America, Pierre. 1987 (June). Inheritance and subtyping in a parallel object-oriented language. In Bezivin, Jean et al., editors, *ECOOP '87, European Conference on Object-Oriented Programming, Paris, France*, pages 234–242, New York, NY. Lecture Notes in Computer Science, Volume 276.

Ananda, Akkihebba L. and Balasubramaniam Srinivasan. 1991. *Distributed Computing Systems: Concepts and Structures.* IEEE Computer Society Press Reprint Collection. Los Alamitos, California: IEEE Computer Society Press.

Andrews, Gregory R. and Ronald A. Olsson. 1993. *The SR Programming Language: Concurrency in Practice.* Redwood City, CA: The Benjamin/Cummings Publishing Company.

Andrews, Gregory R. and Fred B. Schneider. 1983 (March). Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15(1):3–43.

Andrews, Gregory R., Ronald A. Olsson, Michael Coffin, Irving Elshoff, Kelvin Nilsen, Titus Purdin, and Gregg Townsend. 1988 (January). An overview of the sr language and implementation. *ACM Transactions on Programming Languages and Systems*, 10(1):51–86.

Andrews, Gregory R. 1991a. *Concurrent Programming: Principles and Practice.* The Benjamin/Cummings Publishing Company.

———. 1991b (March). Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90.

American National Standards Institute. 1974. *American National Standard Programming Language COBOL.* New York, NY. ANS X3.23-1974.

Apt, Krzysztof R. and Ernst-Rudiger 0lderog. 1991. Introduction to program verification. In Neuhold, E. J. and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 363–429. New York, NY: Springer-Verlag.

Apt, Krzysztof and Andrea Schaerf. 1998 (September). Alma-O: An imperative language that supports declarative programming. *ACM Transactions on Programming Languages and Systems*, 20(5):1014–1066.

Apt, Krzysztof R. 1990. Logic programming. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 10, pages 493–574. New York, NY: The MIT Press.

Armstrong, J. L., M. C. Williams, C. Wikström, and S. R. Virding. 1995. *Concurrent Programming in Erlang.* 2nd edition edition. Prentice Hall.

Arnold, Ken and James Gosling. 1998. *The Java Programming Language*. The Java Series. Second edition. Reading, MA: Addison-Wesley.

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

Astesiano, Edigio. 1991. Inductive and operational semantics. In Neuhold, E. J. and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 51–136. New York, NY: Springer-Verlag.

Backhouse, Roland, Paul Chisholm, Grant Malcolm, and Erik Saaman. 1989 (January – March). Do-it-yourself type theory. *Formal Aspects of Computing*, 1(1):19–84.

Backhouse, R. C. 1989. Constructive type theory – an introduction. In Broy, Manfred, editor, *Constructive Methods in Computing Science*, Vol. F55 of *NATO ASI Series*, pages 9–60. New York, NY: Springer-Verlag.

Backus, J. W. and W. P. Heising. 1964. FORTRAN. *IEEE Transactions on Electronic Computers*, EC-13(4):382–385.

Backus, John. 1978a (August). 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.

———. 1978b (August). The history of FORTRAN I, II, and III. *ACM SIGPLAN Notices*, 13(8):165–180.

Baker, Henry G. 1991 (August). Lively linear lisp — 'look ma, no garbage!'. *ACM SIGPLAN Notices*, 27(8):89–98.

Bal, Henri E., Jennifer G. Steiner, and Andrew S. Tanenbaum. 1989 (September). Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322.

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

———. 1990. Functional programming and lambda calculus. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 7, pages 321–363. New York, NY: The MIT Press.

Barnes, John, Ben Brosgol, et al. 1995 (January). Ada 95 rationale. the language. the standard libraries. Technical report, Intermetrics Inc., 733 Concord Av, Cambridge, MA 02138.

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

Bergstra, J. A., J. Heering, and P. Klint. 1989. *Algebraic Specification*. ACM Press and Addison-Wesley.

Bird, Richard J. and Philip Wadler. 1988. *Introduction to Functional Programming*. International Series in Computer Science. New York, NY: Prentice-Hall.

Black, Andrew P. and Norman C. Hutchinson. 1990 (December). Typechecking polymorphism in emerald. Technical Report TR 90-34, Department of Computer Science; The University of Arizona, Tucson, AZ 85721.

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

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

Black, Andrew, Norman Hutchinson, Eric Jul, Henry Levy, and Larry Carter. 1987 (January). Distribution and abstract types in emerald. *IEEE Transactions on Software Engineering*, SE-13(1):65–76.

Blair, Gordon, John Gallagher, David Hutchison, and Doug Shepherd, editors. 1991. *Object-Oriented Languages, Systems and Applications*. London: Pitman Publishing. ISBN 0-273-03132-5.

Bloom, Bard and Albert R. Meyer. 1990. Experimenting with process equivalence. In Kwiatkowska, M. Z., M. W. Shields, and R. M. Thomas, editors, *Semantics for Concurrency, Leicester*, Workshops in Computing, pages 81–95. New York, NY: Springer-Verlag.

Boehm, Hans-J. 1989 (July). Type inference in the presence of type abstraction. *ACM SIGPLAN Notices*, 24(7):192–206. Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implementation, Portland, Oregon, June.

Booch, Grady. 1991. *Object-Oriented Design: With Applications*. New York, NY: Benjamin Cummings.

Boyland, John and Giuseppe Castagna. 1997 (October). Parasitic methods: Implementation of multi-methods for Java. In *Conference Proceedings of OOPSLA '97, Atlanta*, Vol. 32(10) of *ACM SIGPLAN Notices*, pages 66–76. ACM.

Bracha, Gilad and William Cook. 1990 (October). Mixin-based inheritance. *ACM SIGPLAN Notices*, 25(10):303–311. *OOPSLA ECOOP '90 Proceedings*, N. Meyrowitz (editor).

Bracha, Gilad, Martin Odersky, David Stoutamire, and Philip Wadler. 1998 (October). Making the future safe for the past: Adding genericity to the Java programming language. In *OOPSLA '98 Conference Proceedings*, Vol. 33(10) of *ACM SIGPLAN Notices*, pages 183–200.

Brand, P., S. Haridi, and D.H.D. Warren. 1989. Andorra prolog. *New Generation Computing*, 7(2-3):109–129.

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

Briot, Jean-Pierre, Rachid Guerraoui, and Klaus-Peter Lohr. 1998 (September). Concurrency and distribution in object-oriented programming. *ACM Computing Surveys*, 30(3):291–329.

Brookes, S. D., C. A. R. Hoare, and A. W. Roscoe. 1984 (July). A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599.

Brown, J. A. and H. P. Crowder. 1991. APL2: Getting started. *IBM Systems Journal*, 30(4):433–445.

Brown, James A. 1988. *APL2 at a glance*. Englewood Cliffs, N.J.: Prentice Hall.

Broy, Manfred, Martin Wirsing, and Petter Pepper. 1987 (January). On the algebraic definition of programming languages. *ACM Transactions on Programming Languages and Systems*, 9(1):54–99.

Broy, Manfred. 1991. Formalization of distributed, concurrent, reactive systems. In Neuhold, E. J. and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 319–361. New York, NY: Springer-Verlag.

Bruce, Kim and John C. Mitchell. 1992 (January). 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.

Bruce, Kim B. and Peter Wegner. 1990 (August). An algebraic model of subtype and inheritance. In Bançilhon, Francois and Peter Buneman, editors, *Advances in Database Programming Languages*, pages 75–96. Reading, Mass.: Addison-Wesley.

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

Bruce, Kim, Luca Cardelli, Giuseppe Castagna, The Hopkins Object Group, Gary T. Leavens, and Benjamin Pierce. 1995. On binary methods. *Theory and Practice of Object Systems*, 1(3):221–242.

Bruce, Kim B., Luca Cardelli, and Benjamin C. Pierce. 1996 (July). Comparing object encodings. In *Invited lecture at Third Workshop on Foundations of Object Oriented Languages (FOOL 3)*. Available electronically through `http://www.cs.williams.edu/~kim/FOOL/Abstracts.html`.

Bruce, K. 1993. Safe type checking in a statically typed object-oriented programming language. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 285–298.

Bruce, K. B. 1994 (April). A paradigmatic object-oriented programming language: Design, static typing and semantics. *Journal of Functional Programming*, 4(2):127–206.

Budd, Timothy. 1991a (January). Blending imperative and relational programming. *IEEE Software*, 8(1):58–65.

———. 1991b. *Object-Oriented Programming*. New York, NY: Addison-Wesley.

Budd, Timothy A. 1995. *Multiparadigm Programming in LEDA*. New York, NY: Addison-Wesley.

Butcher, Paul and Hussein Zedan. 1991 (August). Lucinda–an overview. *ACM SIGPLAN Notices*, 26(8):90–100.

———. 1992. Lucinda - a polymorphic linda. In Banatre, J. P. and D. Le Metayer, editors, *Research Directions in High-Level Parallel Programming Languages, Mont Saint-Michel, France, June 1991, Proceedings*, Vol. 574 of *Lecture Notes in Computer Science*, pages 126–146. New York, NY: Springer-Verlag.

Canning, Peter, William Cook, Walter Hill, John Mitchell, and Walter Olthoff. 1989 (September). F-bounded polymorphism for object-oriented programming. In *Fourth International Conference on Functional Programming and Computer Architecture*. ACM. Also technical report STL-89-5, from Software Technology Laboratory, Hewlett-Packard Laboratories.

Cardelli, Luca and Xavier Leroy. 1990 (March). Abstract types and the dot notation. Technical Report 56, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301. Order from src-report@src.dec.com.

Cardelli, Luca and Peter Wegner. 1985 (December). On understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, 17(4):471–522.

Cardelli, Luca, Jim Donahue, Mick Jordan, Bill Kalsow, and Greg Nelson. 1989 (January). The modula-3 type system. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas*, pages 202–212. ACM.

Cardelli, Luca, Simone Martini, John C. Mitchell, and Andre Scedrov. 1994 (Feb). An extension of system f with subtyping. *Information and Computation*, 109(1/2):4–56.

Cardelli, Luca. 1987 (April). Basic polymorphic typechecking. *Science of Computer Programming*, 8(2):147–172.

———. 1988a (February/March). A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164. A revised version of the paper that appeared in the 1984 Semantics of Data Types Symposium, LNCS 173, pages 51–66.

———. 1988b (January). 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.

———. 1991. Typeful programming. In Neuhold, E. J. and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 431–507. New York, NY: Springer-Verlag.

———. 1995 (January). A language with distributed scope. *Computing Systems*, 8(1):27–59. A preliminary version appeared in POPL '95.

———. 1997 (January). Program fragments, linking, and modularization. In *Conference Record of POPL 97: The 24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France*, pages 266–277–, New York, NY.

Carriero, Nicholas and David Gelernter. 1989 (September). How to write parallel programs: A guide for the perplexed. *ACM Computing Surveys*, 21(3):323–357.

Cartwright, Robert and Guy L. Steele Jr. 1998 (October). Compatible genericity with run-time types for the Java programming language. In *OOPSLA '98 Conference Proceedings*, Vol. 33(10) of *ACM SIGPLAN Notices*, pages 201–215.

Castagna, Giuseppe, Giorgio Ghelli, and Giuseppe Longo. 1992 (June). A calculus for overloaded functions with subtyping. In *ACM Conference on LISP and Functional Programming*, pages 182–192. ACM. To appear in *Information and Computation*.

———. 1995 (February). A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135. A preliminary version appeared in *ACM Conference on LISP and Functional Programming*, June 1992 (pp. 182–192).

Castagna, G. 1993 (October). A meta-language for typed object-oriented languages. In Shyamasundar, R. K., editor, *Foundations of Software Technology and Theoretical Computer Science*, Vol. 761 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag.

Castagna, Giuseppe. 1995. Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447.

———. 1997. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science. Boston: Birkhauser.

Chambers, Craig and Gary T. Leavens. 1995 (November). Typechecking and modules for multi-methods. *TOPLAS*, 17(6):805–843.

Chambers, Craig. 1992. Object-oriented multi-methods in Cecil. In Madsen, Ole Lehrmann, editor, *ECOOP '92, European Conference on Object-Oriented Programming, Utrecht, The Netherlands*, Vol. 615 of *Lecture Notes in Computer Science*, pages 33–56. New York, NY: Springer-Verlag.

Chen, Chih-Ping and Paul Hudak. 1997 (January). Rolling your own mutable ADT — a connection between linear types and monads —. In *Conference Record of POPL 97: The 24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France*, pages 54–66, New York, NY.

Chin, Roger S. and Samuel T. Chanson. 1991 (March). Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):91–124.

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

Clocksin, W. F. and C. S. Mellish. 1981. *Programming in Prolog*. New York, NY: Springer-Verlag.

Cohen, Edward. 1990. *Programming in the 1990s: An Introduction to the Calculation of Programs*. New York, NY: Springer-Verlag.

Constable, Robert L. and Daniel R. Zlatin. 1984 (January). The type theory of pl/cv3. *ACM Transactions on Programming Languages and Systems*, 6(1):94–117.

Constable, Robert L. 1989. Assigning meaing to proofs: a semantic basis for problem solving environments. In Broy, Manfred, editor, *Constructive Methods in Computing Science*, Vol. F55 of *NATO ASI Series*, pages 63–91. New York, NY: Springer-Verlag.

Cook, William and Jens Palsberg. 1989 (October). A denotational semantics of inheritance and its correctness. *ACM SIGPLAN Notices*, 24(10):433–443. OOPSLA '89 Conference Proceedings, Norman Meyerowitz (editor), October 1989, New Orleans, Louisiana.

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

Cook, W. R. 1989 (August). A proposal for making eiffel type-safe. *The Computer Journal*, 32(4):305–311.

Cook, William R. 1991. Object-oriented programming versus abstract data types. In de Bakker, J. W., W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, Vol. 489 of *Lecture Notes in Computer Science*, pages 151–178. New York, NY: Springer-Verlag.

Cook, W. R. 1992 (October). Interfaces and specifications for the Smalltalk-80 collection classes. *ACM SIGPLAN Notices*, 27(10):1–15. *OOPSLA '92 Proceedings*, Andreas Paepcke (editor).

Coquand, Thierry and Gérard Huet. 1988 (February/March). The calculus of constructions. *Information and Computation*, 76(2/3):95–120.

Cormack, G. V. and A. K. Wright. 1990 (June). Type-dependent parameter inference. *ACM SIGPLAN Notices*, 25(6):127–136. Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation, White Plains, NY.

Cousot, Patrick. 1990. Methods and logics for proving programs. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 15, pages 841–993. New York, NY: The MIT Press.

———. 1997 (January). Types as abstract interpretation. In *Conference Record of POPL 97: The 24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France*, pages 316–331, New York, NY.

Cox, Brad J. 1986. *Object Oriented Programming: an Evolutionary Approach.* Reading, Mass.: Addison-Wesley Publishing Co.

Danforth, Scott and Chris Tomlinson. 1988 (March). Type theories and object-oriented programming. *ACM Computing Surveys*, 20(1):29–72.

Davie, Anthony J. T. 1992. *An Introduction to Functional Programming Systems Using Haskell.* Cambridge Computer Science Texts. New York, NY: Cambridge University Press.

de Bruijn, N. G. 1980. A survey of the project automath. In Seldin, J. P. and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. New York, NY: Academic Press, Inc.

de Champeaux, Dennis and Penelope Faure. 1992 (March). A comparative study of object-oriented analysis methods. *Journal of Object-Oriented Programming*, 5(1):21–33.

de Champeaux, Dennis, Doug Lea, and Penelope Faure. 1992 (October). The process of object-oriented design. *ACM SIGPLAN Notices*, 27(10):45–62. *OOPSLA '92 Proceedings*, Andreas Papecke (editor).

———. 1993. *Object Oriented System Development.* Mass,: Addison-Wesley Publishing Co.

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

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

Donahue, James and Alan Demers. 1985 (July). Data types are values. *ACM Transactions on Programming Languages and Systems*, 7(3):426–445.

Drossopoulou, Sophia, Susan Eisenbach, and Sarfraz Khurshid. 1999. Is the java type system sound? *Theory and Practice of Object Systems*, 5(1):3–24.

Dybjer, Peter. 1990 (June). Comparing integrated and external logics of functional programs. *Science of Computer Programming*, 14(1):59–79.

Eifrig, J., S. Smith, and V. Trifonov. 1995. Sound polymorphic type inference for objects. In *OOPSLA '95 Conference Proceedings*, Vol. 30(10) of *ACM SIGPLAN Notices*, pages 169–184.

Ellis, Margaret A. and Bjarne Stroustrup. 1990. *The Annotated C++ Reference Manual.* Reading, Mass.: Addison-Wesley Publishing Co.

Felleisen, Matthias. 1990 (May). On the expressive power of programming languages. In Jones, N., editor, *ESOP '90 3rd European Symposium on Programming, Copenhagen, Denmark*, Vol. 432 of *Lecture Notes in Computer Science*, pages 134–151. New York, NY: Springer-Verlag.

Findler, Robert Bruce and Matthew Flatt. 1999 (June). Modular object-oriented programming with units and mixins. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Vol. 34(1) of *ACM SIGPLAN Notices*, pages 94–104. ACM.

Fischer, Kathleen and John C. Mitchell. 1998. On the relationship between classes, objects and data abstraction. *Theory and Practice of Object Systems*, 4(1):3–25.

Fischer, Kathleen and John Reppy. 1999 (May). The design of a class mechanism for Moby. *ACM SIGPLAN Notices*, 34(5):37–49. Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).

Flon, Lawrence. 1975 (October). On research in structured programming. *ACM SIGPLAN Notices*, 10(10):16–17.

Floyd, Robert W. 1979 (August). The paradigms of programming. *Communications of the ACM*, 22(8):455–460.

Friedman, Daniel P., Mitchell Wand, and Christopher T. Haynes. 1992. *Essentials of Programming Languages.* New York, NY: McGraw-Hill Book Co.

Futatsugi, Kokichi, Joseph A. Goguen, Jean-Pierre Jouannaud, and Jose Meseguer. 1985 (January). Principles of OBJ2. In *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 52–66. ACM.

Gabriel, Richard P. 1993 (July). The end of history and the last programming language. *Journal of Object-Oriented Programming*, 6(4):90–94.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, Mass.: Addison-Wesley.

Gelernter, David and Nicholas Carriero. 1992 (February). Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107.

Ghelli, Giorgio. 1991a. Modelling features of object-oriented languages in second order functional languages with subtypes. In de Bakker, J. W., W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, Vol. 489 of *Lecture Notes in Computer Science*, pages 311–340. New York, NY: Springer-Verlag.

———. 1991b (November). A static type system for message passing. *ACM SIGPLAN Notices*, 26(11):129–145. OOPSLA '91 Conference Proceedings, Andreas Paepcke (editor), October 1991, Phoenix, Arizona.

Ghezzi, Carlo and Mehdi Jazayeri. 1987. *Programming Language Concepts 2/E.* New York, NY: John Wiley and Sons.

Gifford, David K. and John M. Lucassen. 1986 (August). Integrating functional and imperative programming. In *ACM Conference on LISP and Functional Programming*, pages 28–38. ACM.

Girard, Jean-Yves, Yves Lafont, and Paul Taylor. 1989. *Proofs and Types.* New York, NY: Cambridge University Press.

Girard, Jean-Yves. 1971. 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.

Girard, J. Y. 1986. The system **f** of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192.

Girard, Jean-Yves. 1993. Linear logic: A survey. In Bauer, F. L., W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, Vol. 94 of *NATO ASI Series. Series F : Computer and System Sciences*, pages 63–112. New York, NY: Springer-Verlag.

Glew, Neal and Greg Morrisett. 1999 (January). Type-safe linking and modular assembly language. In *Conference Record of POPL 99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas*, pages 250–261, New York, NY.

Goguen, Joseph A. and José Meseguer. 1986. Eqlog: Equality, types, and generic modules for logic programming. In DeGroot, Douglas and Gary Lindstrom, editors, *Functional and Logic Programming*, pages 295–363. Englewood Cliffs, N.J.: Prentice-Hall, Inc. An earlier version appeared in the *Journal of Logic Programming*, 1984, Volume 1, Number 2, Pages 179-209.

———. 1987. Unifying functional, object-oriented and relational programming with logical semantics. In Shriver, Bruce and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. Cambridge, Mass.: The MIT Press.

Goguen, Joseph A. 1984 (September). Parameterized programming. *IEEE Transactions on Software Engineering*, SE-10(5):528–543.

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

Goldblatt, R. 1984. *Studies in Logic and the Foundations of Mathematics.* Volume 98, *Topoi: The Categorical Analysis of Logic (Revised Edition).* New York, NY: North-Holland.

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

Gordon, Michael J. C. 1979. *The Denotational Description of Programming Languages.* New York, NY: Springer-Verlag.

———. 1988. *Programming Language Theory and its Implementation.* Prentice Hall International Series in Computer Science. New York, NY: Prentice-Hall, Inc.

Gosling, James, Bill Joy, and Guy Steele. 1996. *The Java Language Specification*. The Java Series. Reading, MA: Addison-Wesley.

Gray, Jim and Andreas Reuter. 1993. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman.

Gries, David and Fred B. Schneider. 1994. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. New York, NY: Springer-Verlag.

Gries, David. 1981. *The Science of Programming*. New York, NY: Springer-Verlag.

Grimshaw, Andrew S. 1993 (May). Easy-to-use object-oriented parallel processing with mentat. *IEEE Computer*, 26(5):39–51.

Griswold, Ralph E. and Madge T. Griswold. 1983. *The Icon Programming Language*. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Griswold, R. E., J. F. Poage, and I. P. Polonsky. 1971. *The SNOBOL4 Programming Language (second edition)*. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

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

Habermann, A. N. 1973. Critical comments on the programming language pascal. *Acta Informatica*, 3(1):47–57.

Hanson, David R. and Ralph E. Griswold. 1978 (May). The sl5 procedure mechanism. *Communications of the ACM*, 21(5):392–400.

Harland, D. M. 1984. *Polymorphic Programming Languages: Design and Implementation*. New York, NY: John Wiley and Sons.

Harper, Robert and Mark Lillibridge. 1994 (January). A type-theoretic approach to higher-order modules with sharing. In *Conference Record of POPL '94: 21ST ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*, pages 123–137, New York, NY.

Harper, Robert. 1994. A simplified account of polymorphic references. *Information Processing Letters*, 51:201–206.

Hassan, Saniya Ben, Henri E. Bal, and Ceriel J. H. Jacobs. 1998 (November). A task- and data-parallel programming language based on shared objects. *ACM Transactions on Programming Languages and Systems*, 20(6):1131–1170.

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

Henglein, Fritz. 1999. Breaking through the n^3 barrier: Faster object type inference. *Theory and Practice of Object Systems*, 5(1):57–72.

Hennessy, Matthew. 1988. *Algebraic Theory of Processes*. Cambridge, Mass.: The MIT Press.

———. 1990. *The Semantics of Programming Languages: an Elementary Introduction using Structural Operational Semantics*. New York, NY: John Wiley and Sons.

Henson, Martin C. 1987. *Elements of Functional Languages*. Oxford, England: Blackwell Scientific Publications.

Hesselink, Wim H. 1992. *Cambridge Tracts in Theoretical Computer Science*. Volume 27, *Programs, Recursion, and Unbounded Choice*. New York, NY: Cambridge University Press.

High Order Language Working Group, Department of Defense. 1978 (June). Department of defense requirements for high order computer programming languages: Steelman. Technical report, U. S. Department of Defense.

Hillis, W. Daniel and Guy L. Steele Jr. 1986 (December). Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183.

Hoang, My and John C. Mitchell. 1995 (January). Lower bounds on type inference with subtypes. In *Conference Record of POPL '95: 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, Calif.*, pages 176–185, New York, NY.

Hoare, C. A. R. and N. Wirth. 1973. An axiomatic definition of the programming language pascal. *Acta Informatica*, 2(4):335–355.

Hoare, C. A. R. 1969 (October). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583.

———. 1978 (August). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677.

———. 1980 (October). Hints on programming language design. In Wasserman, Anthony I., editor, *Tutorial Programming Language Design*, pages 43–52. IEEE.

———. 1985. *Communicating Sequential Processes*. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

———. 1989. Notes on an approach to category theory for computer scientists. In Broy, Manfred, editor, *Constructive Methods in Computing Science*, Vol. F55 of *NATO ASI Series*, pages 245–305. New York, NY: Springer-Verlag.

Horowitz, Ellis. 1987. *Programming Languages: A Grand Tour (Third Edition)*. Rockville, Maryland: Computer Science Press.

Howard, W. A. 1980. The formulae-as-types notion of construction. In Seldin, J. P. and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. New York, NY: Academic Press, Inc.

Hudak, Paul and Joseph H. Fasel. 1992 (May). A gentle introduction to Haskell. *ACM SIGPLAN Notices*, 27(5).

Hudak, Paul, Simon Peyton Jones, Philip Wadler, et al. 1992 (May). Report on the programming language Haskell: A non-strict, purely functional language, version 1.2. *ACM SIGPLAN Notices*, 27(5).

Hudak, Paul. 1989 (September). Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411.

Ichbiah, J. D., J. G. P. Barnes, J. C. Heliard, B. Krieg-Brueckner, O. Roubine, and B. A. Wichmann. 1979 (June). Reference manual and rationale for the ada programming language. *ACM SIGPLAN Notices*, 14(6). This version of the language is now obsolete, but the rationale (part B) is still valuable.

Ichbiah, J., J. Barnes, R. Firth, and M. Woodger. 1991. *Rationale for the Design of the Ada Programming Language*. New York, NY: Cambridge University Press. ISBN 0-521-39267-5.

Ingalls, Dan, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. 1997 (October). Back to the future: The story of squeak, a practical smalltalk written in itself. In *Conference Proceedings of OOPSLA '97, Atlanta*, Vol. 32(10) of *ACM SIGPLAN Notices*, pages 318–326. ACM.

Ingalls, D. H. H. 1981 (August). Design principles behind smalltalk. *BYTE*, 6(8):286–298.

Iverson, K. 1962. *A Programming Language*. New York, NY: John Wiley and Sons.

Iverson, Kenneth E. 1979 (October). Operators. *ACM Transactions on Programming Languages and Systems*, 1(2):161–176.

———. 1991. A personal view of APL. *IBM Systems Journal*, 30(4):582–593.

Jaffar, Joxan, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap. 1992 (July). The clp(∇) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395.

Jayaraman, Bharat and Gopal Gupta. 1989 (June). EqL: The language and its implementation. *IEEE Transactions on Software Engineering*, 15(6):771–779.

Jensen, Kathleen and Niklaus Wirth. 1974. *PASCAL User Manual and Report (second edition)*. New York, NY: Springer-Verlag.

Johnson, Ralph E. 1992 (October). Documenting frameworks using patterns. *ACM SIGPLAN Notices*, 27(10):63–76. *OOPSLA '92 Proceedings*, Andreas Paepcke (editor).

Jones, Mark P. 1995 (Jan). A system of constructor classes: overloading and implicit higher-order polymorphism. *Journal of Functional Programming*, 5(1):1–35. An earlier version appeared in FPCA '93.

Kagawa, Koji. 1997 (August). Compositional references for stateful functional programming. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '97)*, Vol. 32(8) of *ACM SIGPLAN Notices*, pages 217–226. ACM.

Kamin, Samuel. 1988 (January). Inheritance in smalltalk-80: A denotational definition. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, Calif.*, pages 80–87. ACM.

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

Keene, Sonya E. 1989. *Object-Oriented Programming in Common Lisp*. Reading, Mass.: Addison Wesley.

Kelsey, Richard, William Clinger, and Jonathan Rees (Editors). 1998 (September). Revised^5 report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 33(9):26–76.

Kernighan, Brian W. and Dennis M. Ritchie. 1978. *The C Programming Language*. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Kiczales, Gregor, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997 (June). Aspect-oriented programming. In Akşit, Mehmet and Satoshi Matsuoka, editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, Vol. 1241 of *Lecture Notes in Computer Science*, pages 220–242. New York, NY: Springer-Verlag.

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

Kim, Won and Frederick H. Lochovsky, editors. 1989. *Object-Oriented Concepts, Databases, and Applications*. Reading, Mass.: Addison-Wesley Publishing Co.

Klint, Paul. 1993 (April). A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2(2):176–201.

Knuth, Donald E. 1967 (October). The remaining trouble spots in algol 60. *Communications of the ACM*, 10(1):611–617.

Kobayashi, Naoki. 1999 (January). Quasi-linear types. In *Conference Record of POPL 99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas*, pages 29–42, New York, NY.

Kowalski, Robert. 1979 (July). Algorithm = logic + control. *Communications of the ACM*, 22(7):424–435.

LaLonde, Wilf R., Dave A. Thomas, and John R. Pugh. 1986 (November). An exemplar based Smalltalk. *ACM SIGPLAN Notices*, 21(11):322–330. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

LaLonde, Wilf R. 1989 (April). Designing families of data types using exemplars. *ACM Transactions on Programming Languages and Systems*, 11(2):212–248.

Lamport, Leslie and Nancy Lynch. 1990. Distributed computing: Models and methods. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 19, pages 1157–1199. New York, NY: The MIT Press.

Lamport, Leslie and Fred B. Schneider. 1984 (April). The "hoare logic" of csp and all that. *ACM Transactions on Programming Languages and Systems*, 6(2):281–296.

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

———. 1965. A correspondence algol 60 and church's lambda notation. *Communications of the ACM*, 8:89–101, 158–165.

———. 1966 (March). The next 700 programming languages. *Communications of the ACM*, 9(3):157–166.

Lane, Saunders Mac. 1971. *Graduate Texts in Mathematics*. Volume 5, *Categories for the Working Mathematician*. New York, NY: Springer-Verlag.

Launchbury, John and Amr Sabry. 1997 (August). Monadic state: Axiomatization and type safety. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '97)*, Vol. 32(8) of *ACM SIGPLAN Notices*, pages 227–238. ACM.

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

Leavens, Gary T. and Todd D. Millstein. 1998 (October). Multiple dispatch as dispatch on tuples. In *OOPSLA '98 Conference Proceedings*, Vol. 33(10) of *ACM SIGPLAN Notices*, pages 374–387.

Leavens, Gary T. and Don Pigozzi. 1999. Class-based and algebraic models of objects. In Cleaveland, Rance, Michael Mislove, and Philip Mulry, editors, *US—Brazil Joint Workshops on the Formal Foundations of Software Systems*, Vol. 14 of *Electronic Notes in Theoretical Computer Science*. Elsevier. http://www.elsevier.nl/locate/entcs/volume14.html.

Leavens, Gary T. and William E. Weihl. 1990 (October). Reasoning about object-oriented programs that use subtypes (extended abstract). In Meyrowitz, N., editor, *OOPSLA ECOOP '90 Proceedings*, Vol. 25(10) of *ACM SIGPLAN Notices*, pages 212–223. ACM.

———. 1995 (November). Specification and verification of object-oriented programs using supertype abstraction. *Acta Informatica*, 32(8):705–778.

Leavens, Gary T. 1991a (October). Introduction to the literature on object-oriented design, programming, and languages. *OOPS Messenger*, 2(4).

———. 1991b (July). Modular specification and verification of object-oriented programs. *IEEE Software*, 8(4):72–80.

Lecarme, O. and P. Desjardins. 1975. Reply to a paper by a. n. habermann on the programming language pascal. *Acta Informatica*, 4(3):231–243. An earlier version appeared in ACM SIGPLAN Notices, October, 1974.

Lee, Oukseh and Kwangkeun Yi. 1998 (July). Proofs about a folklore let-polymorphic type inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(4):707–723.

Leler, Wm. 1988. *Constraint Programming Languages: Their Specification and Generation*. Reading, Mass.: Addison-Wesley Publishing Co.

Leroy, Xavier. 1994 (January). Manifest types, modules, and separate compilation. In *Conference Record of POPL '94: 21ST ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*, pages 109–122, New York, NY.

Liang, Sheng and Gilad Bracha. 1998 (October). Ownership types for flexible alias protection. In *OOPSLA '98 Conference Proceedings*, Vol. 33(10) of *ACM SIGPLAN Notices*, pages 36–47. ACM.

Liang, Sheng, Paul Hudak, and Mark Jones. 1995 (January). Monad transformers and modular interpreters. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California*, pages 333–343. ACM.

Lieberman, Henry. 1986 (November). Using prototypical objects to implement shared behavior in object oriented systems. *ACM SIGPLAN Notices*, 21(11):214–223. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

Lindsey, C. H. and S. G. van der Meulen. 1977. *Informal Introduction to ALGOL 68 (revised edition)*. New York, NY: North-Holland Publishing Co.

Lindstrom, Gary, Barbara Liskov, and David MacQueen. 1989 (November). Critical research directions in programming languages. *ACM SIGPLAN Notices*, 24(11):10–25.

Lippman, Stanley B. 1991. *C++ Primer: 2nd Edition.* Reading, Mass.: Addison-Wesley Publishing Co.

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

Liskov, Barbara and Robert Scheifler. 1983 (July). Guardians and actions: Linguistic support for robust, distributed programs. *ACM Transactions on Programming Languages and Systems*, 5(3):381–404.

Liskov, Barbara H. and Alan Snyder. 1979 (November). Exception handling in clu. *IEEE Transactions on Software Engineering*, SE-5(6):546–558.

Liskov, Barbara and Jeannette Wing. 1994 (November). A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841.

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

Liskov, Barbara, Russell Atkinson, Toby Bloom, Eliot Moss, J. Craig Schaffert, Robert Scheifler, and Alan Snyder. 1981. *Lecture Notes in Computer Science.* Volume 114, *CLU Reference Manual.* New York, NY: Springer-Verlag.

London, R. L., J. V. Guttag, J. J. Horning, B. W. Lampson, J. G. Mitchell, and G. J. Popek. 1978. Proof rules for the programming language euclid. *Acta Informatica*, 10(1):1–26.

Loveman, D. B. 1993. High performance fortran. *IEEE Parallel and Distributed Technology: Systems and Applications*, 1(1):25–42.

Lucassen, John M. and David K. Gifford. 1988 (January). Polymorphic effect systems. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, Calif.*, pages 47–57. ACM.

Mackie, Ian. 1993. Lilac — a functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):395–433.

MacLennan, Bruce J. 1999. *Principles of Programming Languages.* Third edition. New York, NY: Oxford University Press.

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

Manes, Ernest G. and Michael A. Arbib. 1986. *Algebraic Approaches to Program Semantics.* New York, NY: Springer-Verlag.

Marti-Oliet, Narciso and Jose Meseguer. 1990 (December). Inclusions and subtypes. Technical Report SRI-CSL-90-16, Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park, Calif.

Martin-Löf, P. 1975. An intuitionistic theory of types: Predictive part. In Rose, H. E. and J. C. Sheperdson, editors, *Logic Colloquium '73*, Vol. 80 of *Studies in Logic*, pages 73–118. New York, NY: North-Holland Publishing Co.

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

Matthews, David C. J. 1985a (August). An overview of the poly programming language. In *Persistence and Data Types: Papers for the Appin Workshop*, pages 265–274. Universities of Glasgow and St. Andrews, Departments of Computer Science. Persistent Programming Research Report 16.

———. 1985b (September). Poly manual. *ACM SIGPLAN Notices*, 20(9):52–76.

McCarthy, John, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hart, and Michael I. Levin. 1965. *LISP 1.5 Programmer's Manual.* Cambridge, Mass.: The MIT Press.

McDermott, Drew V. and Gerald Jay Sussman. 1974 (January). The conniver reference manual. AI Memo 295a, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

McDermott, Drew. 1980 (July). The prolog phenomenon. *ACM SIGART Newsletter*, pages 16–20. Number 72.

McDonald, C. and L. Allison. 1989 (October). Denotational semantics of a command interpreter and their implementation in standard ml. *The Computer Journal*, 32(5):422–431.

Meyer, Bertrand. 1988. *Object-oriented Software Construction.* New York, NY: Prentice Hall.

———. 1990. *Introduction to the Theory of Programming Languages.* International Series in Computer Science. New York, NY: Prentice Hall.

Miller, D., G. Nadathur, F. Pfenning, and A. Scedrov. 1991 (March). Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1-2):125–158.

Miller, Dale. 1989. Abstractions in logic programs. In Odifreddi, P., editor, *Logic and Computer Science*, pages 329–359. Academic Press.

———. 1990. A logic programming language with lambda-abstraction function variables, and simple unification. In Schroeder-Heister, Peter, editor, *Extensions of Logic Programming, International Workshop, Tubingen, FRG, December, 1989*, Vol. 475 of *Lecture Notes in Computer Science*, pages 253–282. New York, NY: Springer-Verlag.

———. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536.

Millstein, Todd and Craig Chambers. 1999 (June). Modular statically typed multimethods. In Guerraoui, Rachid, editor, *ECOOP '99 — Object-Oriented Programming 13th European Conference, Lisbon Portugal*, Vol. 1628 of *Lecture Notes in Computer Science*, pages 279–303. New York, NY: Springer-Verlag.

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

Milner, R., J. Parrow, and D. Walker. 1992. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77.

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

Milner, Robin. 1978 (December). A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375.

———. 1989. *Communication and Concurrency*. International Series in Computer Science. New York, NY: Prentice Hall.

———. 1990. Operational and algebraic semantics of concurrent processes. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 19, pages 1201–1242. New York, NY: The MIT Press.

Mitchell, John C. and Robert Harper. 1988 (January). 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.

Mitchell, John C. and Gordon D. Plotkin. 1985 (January). 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.

Mitchell, John C. 1990a (January). Toward a typed foundation for method specialization and inheritance. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, Calif.*, pages 109–124. ACM.

———. 1990b. Type systems for programming languages. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 8, pages 365–458. New York, NY: North-Holland.

———. 1991 (September). On abstraction and the expressive power of programming languages. In *Conference on Theoretical Aspects of Computer Software, Sendi Japan.*

Moggi, Eugenio. 1990. 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, Edinburgh, EH9 3JZ.

Morgan, Carroll. 1994. *Programming from Specifications: Second Edition*. Hempstead, UK: Prentice Hall International.

Morris, James H., Jr. 1973 (January). Protection in programming languages. *Communications of the ACM*, 16(1):15–21.

Mosses, Peter D. 1990. Denotational semantics. In van Leewen, J., editor, *Handbook of Theoretical Computer Science*, Vol. B: Formal Models and Semantics, Chapter 11, pages 577–631. New York, NY: The MIT Press.

———. 1992. *Cambridge Tracts in Theoretical Computer Science*. Volume 26, *Action Semantics*. New York, NY: Cambridge University Press.

Mullender, Sape, editor. 1993. *Distributed Systems*. second edition. New York, NY: Addison-Wesley.

Mullin, Mark. 1989. *Object Oriented Program Design With Examples in C++*. Reading, Mass.: Addison-Wesley Publishing Co.

Myers, Andrew C., Joseph A. Bank, and Barbara Liskov. 1997 (January). Parameterized types for Java. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 132–145, New York, NY.

Nadathur, G. and D. Miller. 1990 (October). Higher-order horn clauses. *Journal of the ACM*, 37(4):777–814.

Naur, Peter, J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. 1963 (January). Revised report on the algorithmic language algol 60. *Communications of the ACM*, 1(17).

Nelson, Greg. 1991. *Systems Programming with Modula-3*. Prentice-Hall.

Nielson, F. and H.R. Neilson. 1992. *Semantics with Applications - A Formal Introduction*. New York, NY: John Wiley and Sons.

Nipkov, Tobias and Christian Prehofer. 1994 (April). Type reconstruction for type classes. *Journal of Functional Programming*, 5(2):201–224.

Nipkow, Tobias and David von Oheimb. 1998 (January). Java$_{light}$ is type-safe — definitely. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 161–170, New York, NY.

Nordlander, Johan. 1999 (June). Pragmatic subtyping in polymorphic languages. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Vol. 34(1) of *ACM SIGPLAN Notices*, pages 216–227. ACM.

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

Odersky, Martin and Konstantin Läufer. 1996 (January). Putting type annotations to work. In *Conference Record of POPL '96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersberg Beach, Florida*, pages 54–67, New York, NY.

Odersky, Martin and Philip Wadler. 1997 (January). Pizza into Java: Translating theory into practice. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 146–159, New York, NY.

Odersky, Martin, Martin Sulzmann, and Martin Wehr. 1999. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55.

Odersky, Martin. 1999 (June). Programming with variable functions. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Vol. 34(1) of *ACM SIGPLAN Notices*, pages 105–116. ACM.

Okasaki, Chris. 1998. *Purely Functional Data Structures*. Cambridge, UK: Cambridge University Press.

O'Toole, James William and David K. Gifford. 1989 (July). Type reconstruction with first-class polymorphic values. *ACM SIGPLAN Notices*, 24(7):207–217. Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implementation, Portland, Oregon, June.

Paepcke, Andreas. 1993. *Object-Oriented Programming: The CLOS Perspective*. The MIT Press.

Palsberg, Jens and Christina Pavlopoulou. 1998 (January). From polyvariant flow information to intersection and union types. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 197–208, New York, NY.

Palsberg, Jens and Michael I. Schwartzbach. 1994. *Object-Oriented Type Systems*. John Wiley and sons.

Papelis, Yiannis E. and Thomas L. Casavant. 1992 (March). Specification and analysis of parallel/distributed software and systems by using petri nets with transition enabling functions. *IEEE Transactions on Software Engineering*, 18(3):252–261.

Paulson, Laurence C. 1991. *ML for the Working Programmer*. New York, NY: Cambridge University Press.

Peterson, J. L. 1977 (September). Petri nets. *ACM Computing Surveys*, 9(3):221–252.

Peterson, Gerald E., editor. 1987. *Tutorial: Object-Oriented Computing*. Los Angeles, Calif.: IEEE Computer Society Press. Volume 1: concepts; volume 2: implementations.

Peyton Jones, S. L. 1987. *The Implementation of Functional Programming Languages*. Englewood Cliffs, N.J.: Prentice-Hall.

Pfenning, Frank. 1991. Logic programming in the LF logical framework. In Huet, Gérard and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press.

Pfenning, Frank, editor. 1992. *Types in Logic Programming*. Logic Programming Series. Cambridge, Mass.: The MIT Press.

Pierce, Benjamin C. and David N. Turner. 1994 (April). Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–248. A preliminary version appeared in POPL 1993.

———. 1998 (January). Local type inference. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 252–265, New York, NY.

Pierce, Benjamin, Scott Dietzen, and Spiro Michaylov. 1989 (March). Programming in higher-order typed lambda-calculi. Technical Report CMU-CS-89-111, School of Computer Science, Carnegie Mellon University.

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

Pitman, Kent M. 1983 (May). The revised maclisp manual. Technical Report TR-295, Massachusetts Institute of Technology, Laboratory for Computer Science.

Plotkin, G. D. 1977. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255.

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

Pottier, François. 1999 (June). A framework for type inference with subtyping. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Vol. 34(1) of *ACM SIGPLAN Notices*, pages 228–238. ACM.

Prasad, S., A. Giacalone, and P. Mishra. 1990 (July). Operational and algebraic semantics of facile: A symmetric integration of concurrent and functional programming. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, Vol. 443 of *LNCS*, pages 765–780. Springer-Verlag.

Pratt, Terrence W. and Marvin V. Zelkowitz. 1996. *Programming Languages: Design and Implementation*. Third edition edition. Englewood Cliffs, NJ: Prentice-Hall.

Qian, Zhenyu. 1994 (January). Higher-order equational logic programming. In *Conference Record of POPL '94: 21ST ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*, pages 254–267, New York, NY.

Rémy, Didier and Jérôme Vouillon. 1998. Objective ML: An effective object-oriented extension of ML. *Theory and Practice of Object Systems*, 4(1):27–52.

Reppey, John H. 1993 (February). Concurrent programming with events - the concurrent ml manual. Technical report, AT&T Bell Labs. Available by anonymous ftp from research.att.com.

Reppy, John and Jon Riecke. 1996 (May). Simple objects for standard ml. *ACM SIGPLAN Notices*, 31(5):171–180. Proceedings of the 1996 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).

Reynolds, J. C. 1974. Towards a theory of type structure. In *Programming Symposium, Proceedings, Colloque sur la Programmation, Paris, April 1974*, Vol. 19 of *Lecture Notes in Computer Science*, pages 408–425. New York, NY: Springer-Verlag.

Reynolds, John C. 1980 (January). Using category theory to design implicit conversions and generic operators. In Jones, Neil D., editor, *Semantics-Directed Compiler Generation, Proceedings of a Workshop, Aarhus, Denmark*, Vol. 94 of *Lecture Notes in Computer Science*, pages 211–258. Springer-Verlag.

———. 1985 (March). Three approaches to type structure. In Ehrig, Hartmut, 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)*, Vol. 185 of *Lecture Notes in Computer Science*, pages 97–138. New York, NY: Springer-Verlag.

Rinard, Martin C. and Monica S. Lam. 1992 (January). Semantic foundations of jade. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 105–118. ACM.

Rinard, Martin C., Daniel J. Scales, and Monica S. Lam. 1993 (June). Jade: A high level machine independent language for parallel programming. *COMPUTER*, 26(6):28–38.

Roman, Peter J. McCann Gruia-Catalin. 1998 (February). Compositional programming abstractions for mobile computing. *IEEE Transactions on Software Engineering*, 24(2):97–110.

Scedrov, Andre. 1990. A guide to polymorphic types. In Odifreddi, P., editor, *Logic and Computer Science*, Vol. 31 of *APIC Series*, pages 387–420. New York, NY: Academic Press.

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

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

———. 1994. *The Structure of Typed Programming Languages*. Foundations of Computing Series. Cambridge, Mass.: MIT Press.

Schubert, Aleksy. 1998 (January). Second-order unification and type inference for Church-style polymorphism. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 279–288, New York, NY.

Scott, Dana. 1981. Lectures on a mathematical theory of computation. Technical Monograph PRG-19, Oxford University Computing Laboratory, Programming Research Group. 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).

Sebesta, Robert W. 1996. *Concepts of Programming Languages*. Third edition. Redwood City, Calif.: Benjamin/Cummings.

Sethi, Ravi. 1996. *Programming Languages: Concepts and Constructs*. Second edition. Reading, Mass.: Addison-Wesley.

Shapiro, Ehud and David H.D. Warren. 1993 (March). The fifth generation project: Personal perspectives. *Communications of the ACM*, 36(3):46–48.

Shapiro, Ehud. 1989 (September). The family of concurrent logic programming languages. *ACM Computing Surveys*, 21(3):413–510.

Shaw, Mary and William A. Wulf. 1980 (March). Toward relaxing assumptions in languages and their implementations. *ACM SIGPLAN Notices*, 15(3):45–61.

Shaw, Mary, William A. Wulf, and R. L. London. 1977 (August). Abstraction and verification in alphard: Defining and specifying iteration and generators. *Communications of the ACM*, 20(8):553–564.

Shaw, Mary. 1981. *ALPHARD: Form and Content*. New York, NY: Springer-Verlag.

Shriver, Bruce and Peter Wegner, editors. 1987. *Research Directions in Object-Oriented Programming*. Cambridge, Mass.: The MIT Press.

Skillicorn, David B. and Domenico Talia. 1995. *Programming Languages for Parallel Processing*. IEEE Computer Society Press.

———. 1998 (June). Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169.

Slagle, James R. 1974 (October). Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM*, 21(4):622–642.

Slonneger, Kenneth and Barry L. Kurtz. 1995. *Formal Syntax and Semantics of Programming Languages*. New York, NY: Addison-Wesley.

Smetsers, S., E. Barendsen, M. v. Eekelen, and R. Plasmeijer. 1994. Guaranteeing safe destructive updates through a type system with uniqueness information for graphs. *Lecture Notes in Computer Science*, 776:358–379.

Snow, C. R. 1992. *Cambridge Computer Science Texts*. Volume 26, *Concurrent Programming*. New York, NY: Cambridge University Press.

Snyder, Alan. 1986 (November). Encapsulation and inheritance in object-oriented programming languages. *ACM SIGPLAN Notices*, 21(11):38–45. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

Sokolowski, S. 1991. *Applicative High Order Programming*. Chapman and Hall Computing Series. New York, NY: Chapman & Hall Computing.

Springer, George and Daniel P. Friedman. 1989. *Scheme and the Art of Programming*. New York, NY: McGraw-Hill.

Stallman, Richard, David Moon, and Daniel Weinreb. 1984 (June). *Lisp Machine Manual (sixth edition)*. Cambridge, Mass.: Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Stanisfer, Ryan. 1992. *ML Primer*. Englewood Cliffs, NJ: Prentice Hall.

———. 1995. *The Study of Programming Languages*. Englewood Cliffs, NJ: Prentice Hall.

Steele Jr., Guy Lewis and Gerald Jay Sussman. 1978a (May). The art of the interpreter or, the modularity complex (parts zero, one, and two). AI Memo 453, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

———. 1978b (January). Revised report on scheme a dialect of lisp. AI Memo 452, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Steele Jr., Guy L. 1984. *Common LISP: The Language*. Burlington, Mass.: Digital Press.

———. 1990. *Common LISP: The Language*. Second edition. Bedford, Mass.: Digital Press.

Steele, Jr., Guy L. 1994 (January). Building interpreters by composing monads. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*, pages 472–492. ACM.

Sterling, Leon and Ehud Shapiro. 1994. *The Art of Prolog*. Second edition. Cambridge, Mass.: The MIT Press.

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

Stroustrup, Bjarne. 1991. *The C++ Programming Language: Second Edition*. Reading, Mass.: Addison-Wesley Publishing Co.

———. 1997. *The C++ Programming Language: Third Edition*. Reading, Mass.: Addison-Wesley Publishing Co.

Sussman, G. and G. Steele. 1980. Constraints: a language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14:1–39.

Sussman, Gerald Jay, Terry Winograd, and Eugene Charniak. 1971 (December). Micro-planner reference manual. AI Memo 203A, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Symbolics, Inc. 1984 (March). *Lisp Machine Manual.* Cambridge, Mass. Eight volumes.

Taivalsaari, Antero. 1996 (September). On the notion of inheritance. *ACM Computing Surveys*, 28(3):438–479.

Talpin, Jean-Pierre and Pierre Jouvelot. 1994 (June). The type and effect discipline. *Information and Computation*, 111(2):245–296.

Tanenbaum, Andrew S., Robert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullender, Jack Jansen, and Guido van Rossum. 1990 (December). Experience with the amoeba distributed operating system. *Communications of the ACM*, 33(12):46–63.

Tanenbaum, Andrew S., M. Frans Kaashoek, and Henri E. Bal. 1992 (August). Parallel programming using shared objects and broadcasting. *Computer*, 25(8):10–19.

Tanenbaum, Andrew S. 1976 (June). A tutorial on algol 68. *ACM Computing Surveys*, 8(2).

Tennent, R. D. 1976 (August). The denotational semantics of programming languages. *Communications of the ACM*, 19:437–453.

———. 1977. Language design methods based on semantic principles. *Acta Informatica*, 8:97–112.

———. 1981. *Principles of Programming Languages.* Englewood Cliffs, N.J.: Prentice-Hall International.

Thompson, Simon. 1991. *Type Theory and Functional Programming.* International Computer Science Series. Addison-Wesley Publishing Co.

Thorn, Tommy. 1997 (September). Programming languages for mobile code. *ACM Computing Surveys*, 29(3):213–239.

Thorup, Kresten Krab. 1997 (June). Genericity in Java with virtual types. In Akşit, Mehmet and Satoshi Matsuoka, editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, Vol. 1241 of *Lecture Notes in Computer Science*, pages 444–471. New York, NY: Springer-Verlag.

Tiuryn, Jerzy. 1990. Type inference probelms: A survey. In Rovan, B., editor, *Mathematical Foundations of Computer Science 1990, Banskà Bystrica, Czechoslovakia*, Vol. 452 of *Lecture Notes in Computer Science*, pages 105–120. New York, NY: Springer-Verlag.

Turner, David A. 1990a. An overview of miranda. In Turner, David A., editor, *Research Topics in Functional Programming*, University of Texas at Austin Year of Programming Series, pages 1–16. New York, NY: Addison-Wesley Publishing Co.

Turner, David A., editor. 1990b. *Research Topics in Functional Programming.* University of Texas at Austin Year of Programming Series. New York, NY: Addison-Wesley Publishing Co.

Ullman, Jeffry D. 1994. *Elements of ML Programming.* Englewood Cliffs, NJ: Prentice Hall.

Ungar, David and Randall B. Smith. 1987 (December). Self: The power of simplicity. *ACM SIGPLAN Notices*, 22(12):227–241. OOPSLA '87 Conference Proceedings, Norman Meyrowitz (editor), October 1987, Orlando, Florida.

Utting, Mark and Ken Robinson. 1992. Modular reasoning in an object-oriented refinement calculus. In Bird, R. S., C. C. Morgan, and J. C. P. Woodcock, editors, *Mathematics of Program Construction, Second International Conference, Oxford, U.K., June/July*, Vol. 669 of *Lecture Notes in Computer Science*, pages 344–367. New York, NY: Springer-Verlag.

van Hentenryck, Pascal, Vijay Saraswat, et al. 1996 (December). Strategic directions in object-oriented programming. *ACM Computing Surveys*, 28(4):701–726.

van Leeuwen, Jan. 1990. *Handbook of Theoretical Computer Science.* Vol. B: Formal Models and Semantics. New York, NY: The MIT Press.

van Roy, Peter, Seif Haridi, Per Brand, Gert Smolka, Michael Mehl, and Ralf Scheidhauer. 1997 (September). Mobile objects in distributed oz. *ACM Transactions on Programming Languages and Systems*, 19(5):804–851.

van Wijngaarden, A., B. J. Mailloux, J. E. L. Peck, C. H. Koster, M. Sintzoff, C. H. Lindsey, L. G. L. T. Meertens, and R. G. Fisker. 1977. Revised report on the algorithmic language algol 68. *ACM SIGPLAN Notices*, 12(5):1–70. This has also been published by Springer-Verlag, New York, N. Y., and in Acta Informatica, volume 5, pages 1-236 (1975).

Wadler, Philip and Stephen Blott. 1989 (January). How to make ad-hoc polymorphism less ad hoc. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas*, pages 60–76. ACM.

Wadler, Philip. 1992 (January). The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–14. ACM.

———. 1996 (June). Lazy versus strict. *ACM Computing Surveys*, 28(2):318–320.

———. 1997 (September). How to declare an imperative. *ACM Computing Surveys*, 29(3):240–263.

———. 1999 (June). The marriage of effects and monads. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Vol. 34(1) of *ACM SIGPLAN Notices*, pages 63–74. ACM.

Walters, R. F. C. 1991. *Cambridge Computer Science Texts*. Volume 28, *Categories and Computer Science*. New York, NY: Cambridge University Press.

Wasserman, Anthony I. 1980. *TUTORIAL Programming Language Design*. Los Alamitos, Calif.: IEEE Computer Society Press. Initally presented at Compsac80, The IEEE Computer Society's Fourth International Computer Software & Applications Conference, October 27-31, 1980. The IEEE catalog number is EHO 164-4.

Watt, D. A. 1986. Executable denotational semantics. *Software: Practice and Experience*, 16(1):13–43.

Watt, David A. 1990. *Programming Language Concepts and Paradigms*. Prentice Hall International Series in Computer Science. New York, NY: Prentice-Hall.

———. 1991. *Programming Language Syntax and Semantics*. Prentice Hall International Series in Computer Science. New York, NY: Prentice-Hall.

Wegbreit, Ben. 1974 (May). The treatment of data types in EL1. *Communications of the ACM*, 17(5):251–264.

Weihl, William E. 1990 (April). Linguistic support for atomic data types. *ACM Transactions on Programming Languages and Systems*, 12(2):178–202.

Weinreb, Daniel and David Moon. 1980 (November). Flavors: Message passing in the lisp machine. AI Memo 602, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Wells, J. B. 1994. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science, Paris*, pages 176–185.

Wexelblat, Richard L., editor. 1978 (June). *Preprints — History of Programming Languages Conference, Los Angeles, California*. ACM. *ACM SIGPLAN Notices*, Volume 13, Number 8, August 1978.

Wexelblat, Richard L., editor. 1993 (March). *ACM SIGPLAN History of Programming Languages Conference (HOPL II), Preprints, Cambridge, MA, USA*. ACM. *ACM SIGPLAN Notices*, Volume 28, Number 3.

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

Wirfs-Brock, Rebecca J. and Ralph E. Johnson. 1990 (September). Surveying current research in object-oriented design. *Communications of the ACM*, 33(9):105–124.

Wirfs-Brock, Rebecca, Brian Wilkerson, and Lauren Wiener. 1990. *Designing Object-Oriented Software*. Englewood Cliffs, NJ 07632: Prentice-Hall.

Wirth, N. and C. A. R. Hoare. 1966 (June). A contribution to the development of algol. *Communications of the ACM*, 9(6):413–432.

Wirth, N. 1971. The programming language pascal. *Acta Informatica*, 1(1):35–63.

Wirth, Niklaus. 1974. On the design of programming languages. In *Information Processing 74*, pages 386–393, New York, NY.

———. 1975 (June). An assessment of the programming language pascal. *IEEE Transactions on Software Engineering*, pages 192–198.

———. 1985. *Programming in Modula-2 (3rd corrected edition)*. New York, NY: Springer-Verlag.

Wirth, N. 1988 (April). Type extensions. *ACM Transactions on Programming Languages and Systems*, 10(2):204–214.

Wyatt, Barbara B., Krishna Kavi, and Steve Hufnagel. 1992 (November). Parallelism in object-oriented languages: A survey. *IEEE Software*, 9(6):56–66.