

- [69] Richard Hamming. *The Art of Doing SCIENCE and Engineering: Learning to Learn*. Gordon and Breach Science Publishers, 1997.
- [70] Seif Haridi and Sverker Janson. Kernel Andorra Prolog and its computation model. In *7th International Conference on Logic Programming*, pages 31–48. The MIT Press, June 1990.
- [71] Seif Haridi, Peter Van Roy, Per Brand, Michael Mehl, Ralf Scheidhauer, and Gert Smolka. Efficient logic variables for distributed computing. *ACM Transactions on Programming Languages and Systems*, May 1999.
- [72] Seif Haridi, Peter Van Roy, Per Brand, and Christian Schulte. Programming languages for distributed applications. *New Generation Computing*, 16(3):223–261, May 1998.
- [73] Seif Haridi, Peter Van Roy, and Gert Smolka. An overview of the design of Distributed Oz. In the *2nd International Symposium on Parallel Symbolic Computation (PASCO 97)*. ACM, July 1997.
- [74] Martin Henz. *Objects for Concurrent Constraint Programming*. Internationale Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA, USA, 1997.
- [75] Martin Henz. *Objects for Concurrent Constraint Programming*, volume 426 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, November 1997.
- [76] Martin Henz. Objects in Oz. Doctoral dissertation, Saarland University, Saarbrücken, Germany, May 1997.
- [77] Martin Henz and Leif Kornstaedt. The Oz notation. Technical report, DFKI and Saarland University, December 1999. Available at <http://www.mozart-oz.org/>.
- [78] Martin Henz, Tobias Müller, and Ka Boon Ng. Figaro: Yet another constraint programming library. In *Workshop on Parallelism and Implementation Technology for Constraint Logic Programming, International Conference on Logic Programming (ICLP 99)*, 1999.
- [79] Martin Henz, Gert Smolka, and Jörg Würtz. Oz – a programming language for multi-agent systems. In *13th International Joint Conference on Artificial Intelligence*, pages 404–409. Morgan Kaufmann, August 1993.
- [80] Martin Henz, Gert Smolka, and Jörg Würtz. Oz—a programming language for multi-agent systems. In Ruzena Bajcsy, editor, *13th International Joint Conference on Artificial Intelligence*, volume 1, pages 404–409, Chambéry, France, 30 August–3 September 1993. Morgan Kaufmann Publishers.

- [81] Martin Henz, Gert Smolka, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In Pascal Van Hentenryck and Vijay Saraswat, editors, *Principles and Practice of Constraint Programming*, pages 29–48, Cambridge, Mass., 1995. The MIT Press.
- [82] Charles Antony Richard Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(10):549–557, October 1974.
- [83] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.
- [84] Bruce K. Holmer, Barton Sano, Michael Carlton, Peter Van Roy, and Alvin M. Despain. Design and analysis of hardware for high performance Prolog. *J. Log. Prog.*, 29:107–139, November 1996.
- [85] Paul Hudak. Conception, evolution, and application of functional programming languages. *Computing Surveys*, 21(3):359–411, September 1989.
- [86] Paul Hudak, John Peterson, and Joseph Fasel. A gentle introduction to Haskell version 98. Available at <http://www.haskell.org/tutorial/>.
- [87] John Hughes. Why Functional Programming Matters. *Computer Journal*, 32(2):98–107, 1989.
- [88] Robert A. Iannucci. *Parallel Machines: Parallel Machine Languages. The Emergence of Hybrid Dataflow Computer Architectures*. Kluwer, Dordrecht, the Netherlands, 1990.
- [89] Daniel H. H. Ingalls. Design principles behind Smalltalk. *Byte*, 6(8):286–298, 1981.
- [90] Joxan Jaffar and Michael Maher. Constraint logic programming: A survey. *J. Log. Prog.*, 19/20:503–581, May/July 1994.
- [91] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley Professional Computing, 1991.
- [92] Sverker Janson. *AKL—A Multiparadigm Programming Language*. PhD thesis, Uppsala University and SICS, 1994.
- [93] Sverker Janson and Seif Haridi. Programming paradigms of the Andorra Kernel Language. In *International Symposium on Logic Programming*, pages 167–183, October 1991.
- [94] K. Jensen and N. Wirth. *Pascal: User Manual and Report (Second Edition)*. Springer-Verlag, 1978.
- [95] Richard Jones and Rafael Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley & Sons, 1996.

- [96] Andreas Kågedal, Peter Van Roy, and Bruno Dumant. Logical State Threads 0.1, January 1997. Available at <http://www.info.ucl.ac.be/people/PVR/implementation.html>.
- [97] Gilles Kahn. The semantics of a simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.
- [98] Gilles Kahn and David B. MacQueen. Coroutines and networks of parallel processes. In *IFIP Congress*, pages 993–998, 1977.
- [99] B. W. Kernighan and D. M. Ritchie. *The C Programming Language (ANSI C), Second Edition*. Prentice Hall, 1988.
- [100] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, 1991.
- [101] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley.
- [102] Donald E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, 1973.
- [103] Donald E. Knuth. Structured programming with **go to** statements. *Computing Surveys*, 6(4), December 1974.
- [104] Leif Kornstaedt. Gump – a front-end generator for Oz. Technical report, Mozart Consortium, December 2001. Available at <http://www.mozart-oz.org/>.
- [105] S. Rao Kosaraju. Analysis of structured programs. *J. Computer and System Sciences*, 9(3), December 1974.
- [106] Robert Kowalski. *Logic for Problem Solving*. North-Holland, 1979.
- [107] James F. Kurose and Keith W. Ross. *Computer networking: a top-down approach featuring the Internet*. Addison-Wesley, 2001.
- [108] Leslie Lamport. *LaTeX: A Document Preparation System, Second Edition*. Addison-Wesley, 1994.
- [109] Hugh C. Lauer and Roger M. Needham. On the duality of operating system structures. In *Second International Symposium on Operating Systems, IRIA*, October 1978. Reprinted in *Operating Systems Review*, 13(2), April 1979, pp. 3–19.
- [110] Doug Lea. *Concurrent Programming in Java*. Addison-Wesley, 1997.
- [111] Doug Lea. *Concurrent Programming in Java, Second Edition*. Addison-Wesley, 2000.

- [112] Nancy Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, July 1993.
- [113] Henry Lieberman. Using prototypical objects to implement shared behavior in object-oriented systems. In *1st Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA 86)*, September 1986. Also in *Object-Oriented Computing*, Gerald Peterson, ed., IEEE Computer Society Press, 1987.
- [114] John Lloyd. *Foundations of Logic Programming, Second Edition*. Springer-Verlag, 1987.
- [115] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, Calif., 1996.
- [116] Bruce J. MacLennan. *Principles of Programming Languages, Second Edition*. Saunders, Harcourt Brace Jovanovich, 1987.
- [117] Michael Maher. Logic semantics for a class of committed-choice programs. In *International Conference on Logic Programming (ICLP 87)*, pages 858–876, May 1987.
- [118] Zohar Manna. *The Mathematical Theory of Computation*. McGraw-Hill, 1974.
- [119] Sun Microsystems. *The Remote Method Invocation Specification*, 1997. Available at <http://www.javasoft.com>.
- [120] John McCarthy. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962.
- [121] Michael Mehl, Christian Schulte, and Gert Smolka. Futures and by-need synchronization for Oz. DRAFT. Available at <http://www.mozart-oz.org/papers/>, May 1998.
- [122] Bertrand Meyer. *Object-Oriented Software Construction, Second Edition*. Prentice Hall, 2000.
- [123] Mark Miller, Marc Stiegler, Tyler Close, Bill Frantz, Ka-Ping Yee, Chip Morningstar, Jonathan Shapiro, and Norm Hardy. E: Open source distributed capabilities, 2001. Available at <http://www.erights.org>.
- [124] Mark Miller, Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Draft available at <http://zesty.ca/capmyths>, 2003.
- [125] Mark S. Miller, Chip Morningstar, and Bill Frantz. Capability-based financial instruments. In *Financial Cryptography 2000*, Anguilla, British West Indies, February 2000.

- [126] Robin Milner, Mads Tofte, and Robert Harper. *Definition of Standard ML*. MIT Press, Cambridge, MA, USA, 1990.
- [127] J. Paul Morrison. *Flow-Based Programming: A New Approach to Application Development*. Van Nostrand Reinhold, 1994.
- [128] Almetwally Mostafa, Iliès Alouini, and Peter Van Roy. Fault tolerant global store module, 2001. Available at <http://www.mozart-oz.org/mogul/info/mostafa/globalstore.html>.
- [129] Mozart Consortium. The Mozart Programming System version 1.2.3, December 2001. Available at <http://www.mozart-oz.org/>.
- [130] Peter Naur. Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 1963.
- [131] Rishiyur S. Nikhil. ID language reference manual version 90.1. Technical Report Memo 284-2, MIT, Computation Structures Group, July 1994.
- [132] Rishiyur S. Nikhil. An overview of the parallel language Id – a foundation for pH, a parallel dialect of Haskell. Technical report, Digital Equipment Corporation, Cambridge Research Laboratory, 1994.
- [133] Rishiyur S. Nikhil and Arvind. *Implicit Parallel Programming in pH*. Morgan Kaufmann, 2001.
- [134] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., 1988.
- [135] Theodore Norvell. Monads for the working Haskell programmer – a short tutorial. Available at <http://www.haskell.org/>.
- [136] Peter Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992.
- [137] K. Nygaard and O. J. Dahl. *The Development of the SIMULA Languages*, pages 439–493. Academic Press, 1981.
- [138] Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.
- [139] Richard A. O’Keefe. *The Craft of Prolog*. The MIT Press, 1990.
- [140] Andreas Paepcke, editor. *Object-Oriented Programming: The CLOS Perspective*. The MIT Press, 1993.
- [141] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. The Harvester Press, 1980.

- [142] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
- [143] David Lorge Parnas. Teaching programming as engineering. In *9th International Conference of Z Users*, volume 967 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995. Reprinted in *Software Fundamentals*, Addison-Wesley, 2001.
- [144] David Lorge Parnas. *Software Fundamentals*. Addison-Wesley, 2001.
- [145] F. Paternò. *Model-based Design and Evaluation of Interactive Applications*. Springer-Verlag, Berlin, 1999.
- [146] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach, Second Edition*. Morgan Kaufmann, 1996.
- [147] Simon Peyton Jones. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. In Tony Hoare, Manfred Broy, and Ralf Steinbruggen, editors, *Engineering theories of software construction*, pages 47–96. IOS Press, 2001. Presented at the 2000 Marktoberdorf Summer School.
- [148] Simon Peyton Jones, editor. *Haskell 98 language and libraries: The revised report*. Cambridge University Press, 2003. Also published as the January 2003 Special Issue of the Journal of Functional Programming.
- [149] Simon Peyton Jones, Andrew Gordon, and Sigbjørn Finne. Concurrent Haskell. In *Principles of Programming Languages (POPL)*, 1996.
- [150] Shari Lawrence Pfleeger. *Software Engineering: The Production of Quality Software, Second Edition*. Macmillan, 1991.
- [151] David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. In the *International Workshop on Memory Management*, volume 986 of *Lecture Notes in Computer Science*, pages 211–249, Berlin, September 1995. Springer-Verlag.
- [152] R. J. Pooley. *An Introduction to Programming in SIMULA*. Blackwell Scientific Publishers, 1987.
- [153] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.
- [154] Roger S. Pressman. *Software Engineering, Sixth Edition*. Addison-Wesley, 2000.

- [155] Mahmoud Rafea, Fredrik Holmgren, Konstantin Popov, Seif Haridi, Stelios Lelis, Petros Kavassalis, and Jakka Sairamesh. Application architecture of the Internet simulation model: Web Word of Mouth (WoM). In *IASTED International Conference on Modelling and Simulation MS2002*, May 2002.
- [156] Eric Raymond. The cathedral and the bazaar, May 1997.
- [157] Juris Reinfelds. Teaching of programming with a programmer's theory of programming. In *Informatics Curricula, Teaching Methods, and Best Practice (ICTEM 2002, IFIP Working Group 3.2 Working Conference)*. Kluwer Academic Publishers, 2002.
- [158] John H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999.
- [159] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language reference manual*. Addison-Wesley, 1999.
- [160] Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 1995.
- [161] Oliver Sacks. *The Man Who Mistook His Wife for a Hat: And Other Clinical Tales*. Touchstone Books, 1998.
- [162] Jakka Sairamesh, Petros Kavassalis, Manolis Marazakis, Christos Nikolaos, and Seif Haridi. Information cities over the Internet: Taxonomy, principles and architecture. In *Digital Communities 2002*, November 2001.
- [163] Vijay A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, 1993.
- [164] Vijay A. Saraswat, Martin C. Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In *Principles of Programming Languages (POPL)*, pages 333–352, 1991.
- [165] Steve Schneider. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley & Sons, 2000.
- [166] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.
- [167] Christian Schulte. Programming constraint inference engines. In Gert Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 519–533, Schloß Hagenberg, Austria, October 1997. Springer-Verlag.
- [168] Christian Schulte. Comparing trailing and copying for constraint programming. In *International Conference on Logic Programming (ICLP 99)*, pages 275–289. The MIT Press, November 1999.

- [169] Christian Schulte. *Programming Constraint Inference Services*. PhD thesis, Saarland University, Fachbereich Informatik, Saarbrücken, Germany, 2000.
- [170] Christian Schulte. Programming deep concurrent constraint combinators. In Enrico Pontelli and Vítor Santos Costa, editors, *Practical Aspects of Declarative Languages, Second International Workshop, PADL 2000*, volume 1753 of *Lecture Notes in Computer Science*, pages 215–229, Boston, MA, USA, January 2000. Springer-Verlag.
- [171] Christian Schulte. Oz Explorer – visual constraint programming support. Technical report, Mozart Consortium, December 2001. Available at <http://www.mozart-oz.org/>.
- [172] Christian Schulte. *Programming Constraint Services: High-Level Programming of Standard and New Constraint Services*, volume 2302 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
- [173] Christian Schulte and Gert Smolka. Encapsulated search for higher-order concurrent constraint programming. In *1994 International Symposium on Logic Programming*, pages 505–520. The MIT Press, November 1994.
- [174] Christian Schulte and Gert Smolka. Finite domain constraint programming in Oz. A tutorial. Technical report, DFKI and Saarland University, December 1999. Available at <http://www.mozart-oz.org/>.
- [175] Ehud Shapiro. A subset of Concurrent Prolog and its interpreter. Technical Report TR-003, Institute for New Generation Computer Technology (ICOT), Cambridge, Mass., January 1983.
- [176] Ehud Shapiro, editor. *Concurrent Prolog: Collected Papers*, volume 1-2. The MIT Press, Cambridge, Mass., 1987.
- [177] Ehud Shapiro. The family of concurrent logic programming languages. *ACM Computing Surveys*, 21(3):413–510, September 1989.
- [178] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples*. McGraw-Hill Book Company, 1982.
- [179] Gert Smolka. The definition of Kernel Oz. In Andreas Podelski, editor, *Constraints: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 251–292. Springer-Verlag, Berlin, 1995.
- [180] Gert Smolka. The Oz programming model. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 324–343. Springer-Verlag, Berlin, 1995.
- [181] Guy L. Steele, Jr. *Common LISP: The Language*. Digital Press, 1984.

- [182] Leon Sterling and Ehud Shapiro. *The Art of Prolog—Advanced Programming Techniques*. Series in Logic Programming. The MIT Press, 1986.
- [183] Marc Stiegler. *The E Language in a Walnut*. 2000. Draft, available at <http://www.erights.org>.
- [184] Bjarne Stroustrup. *The C++ Programming Language, Third Edition*. Addison-Wesley, 1997.
- [185] Giancarlo Succi and Michele Marchesi. *Extreme Programming Examined*. Addison-Wesley, 2001.
- [186] Sun Microsystems. *The Java Series*. Sun Microsystems, Mountain View, Calif., 1996. Available at <http://www.javasoft.com>.
- [187] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley and ACM Press, 1999.
- [188] Andrew Taylor. *High-Performance Prolog Implementation*. PhD thesis, Basser Department of Computer Science, University of Sydney, June 1991.
- [189] Gerard Tel. *An Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, United Kingdom, 1994.
- [190] Evan Tick. The deevolution of concurrent logic programming. *J. Log. Prog.*, 23(2):89–123, May 1995.
- [191] Kasunori Ueda. Guarded Horn Clauses. In Eiti Wada, editor, *Proceedings of the 4th Conference on Logic Programming*, volume 221 of *Lecture Notes in Computer Science*, pages 168–179, Tokyo, Japan, July 1985. Springer.
- [192] Jeffrey D. Ullman. *Elements of ML Programming*. Prentice Hall, 1998.
- [193] Peter Van Roy. VLSI-BAM Diagnostic Generator, 1989. Prolog program to generate assembly language diagnostics. Aquarius Project, University of California, Berkeley.
- [194] Peter Van Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* PhD thesis, Computer Science Division, University of California at Berkeley, December 1990. Technical Report UCB/CSD 90/600.
- [195] Peter Van Roy. 1983–1993: The wonder years of sequential Prolog implementation. *J. Log. Prog.*, 19/20:385–441, May/July 1994.
- [196] Peter Van Roy, Per Brand, Denys Duchier, Seif Haridi, Martin Henz, and Christian Schulte. Logic programming in the context of multiparadigm programming: the Oz experience. *Theory and Practice of Logic Programming*, 2003. To appear.

- [197] Peter Van Roy, Per Brand, Seif Haridi, and Raphaël Collet. A lightweight reliable object migration protocol. In Henri E. Bal, Boumediene Belkhouche, and Luca Cardelli, editors, *Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*. Springer Verlag, October 1999.
- [198] Peter Van Roy and Alvin Despain. High-performance logic programming with the Aquarius Prolog compiler. *IEEE Computer*, pages 54–68, January 1992.
- [199] Peter Van Roy and Seif Haridi. Teaching programming broadly and deeply: the kernel language approach. In *Informatics Curricula, Teaching Methods, and Best Practice (ICTEM 2002, IFIP Working Group 3.2 Working Conference)*. Kluwer Academic Publishers, 2002.
- [200] Peter Van Roy and Seif Haridi. Teaching programming with the kernel language approach. In *Workshop on Functional and Declarative Programming in Education (FDPE02), at Principles, Logics, and Implementations of High-Level Programming Languages (PLI2002)*. University of Kiel, Germany, October 2002.
- [201] Peter Van Roy, Seif Haridi, Per Brand, Gert Smolka, Michael Mehl, and Ralf Scheidhauer. Mobile objects in Distributed Oz. *ACM Transactions on Programming Languages and Systems*, 19(5):804–851, September 1997.
- [202] Arthur H. Veen. Dataflow machine architecture. *ACM Computing Surveys*, 18(4):365–396, December 1986.
- [203] Duncan J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, 1999.
- [204] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.
- [205] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.
- [206] Claes Wikström. Distributed programming in Erlang. In the *1st International Symposium on Parallel Symbolic Computation (PASCO 94)*, pages 412–421, Singapore, September 1994. World Scientific.
- [207] Herbert S. Wilf. *generatingfunctionology*. Academic Press, 1994.
- [208] Glynn Winskel. *The Formal Semantics of Programming Languages*. Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts, 1993.

- [209] Noel Winstanley. What the hell are Monads?, 1999. Available at <http://www.haskell.org>.
- [210] David Wood. Use of objects and agents at Symbian, September 2000. Talk given at the Newcastle Seminar on the Teaching of Computing Science.
- [211] Matthias Zenger and Martin Odersky. Implementing extensible compilers. In *1st International Workshop on Multiparadigm Programming with Object-Oriented Languages*, pages 61–80. John von Neumann Institute for Computing (NIC), June 2001. Workshop held as part of ECOOP 2001.

Index

- ! (escaped variable marker), 506, 515
- ! (cut) operation (in Prolog), 675, 679, 682
- \$ (nesting marker), 54, 85, 363, 373
- * (multiplication) operation, 56, 823
- + (addition) operation, 56, 823
- (subtraction) operation, 56, 823
- . (field selector) operation, 56, 828
- . := (dictionary/array assignment) statement, 439, 440, 841
- . := (dictionary/array exchange) expression, 841
- / (floating division) operation, 56, 824
- := (state assignment) statement, 499, 502
- := (state exchange) expression, 502
- = (binding) operation, 45, 47, 48, 101
- == (equality) comparison, 57
- =< (less or equal) comparison, 57
- ? (output argument), 59, 843
- @ (state access) operation, 499, 502
- # (tupling) constructor, 143, 827, 833
- & (inline character) operator, 822
- < (strictly less) comparison, 57, 832
- <= (optional method argument) operator, 505
- > (strictly greater) comparison, 57
- >= (greater or equal) comparison, 57
- \ (backslash), 823
- \= (inequality) comparison, 57
- ~ (minus sign), 822
- | (list pairing) constructor, 54
- Abelson, Harold, xxxiii, 43
- absolute error, 122
- abstract machine, 43
 - relation to semantic rules, 789
 - substitution-based, 128–129
- abstract syntax tree, 164, 165
- abstraction, xxxii
 - active object, 563
 - class, 498
 - collection, 438
 - collector, 192, 326, 434, 488
 - connector, 326
 - control, 125
 - database, 667
 - encapsulated search, 637
 - hierarchy in object-oriented programming, 552
 - lifecycle, 42
 - Linda (tuple space), 594
 - linguistic, 40–41, 126
 - case** (pattern matching), 793
 - class**, 554
 - conc** (concurrent composition), 283
 - delegation, 520
 - for** loop, 190, 451
 - fun** (function), 85
 - functor** (software component), 227
 - gate** (logic gate), 276
 - local vs. global translation, 846
 - monitor, 601
 - parameter passing, 438
 - protected scope (Java), 574
 - while** loop, 451
 - list comprehension, 307
 - lock, 590, 606
 - loop, 184
 - mailbox, 398
 - monitor, 600
 - pipe of stream objects, 567
 - procedural, 180
 - protector, 326
 - queue, 149, 303
 - replicator, 326
 - software component, 223

- specialization hierarchy, xxxii
- termination detection, 390
- transaction, 608
- tuple space, 594
- access (cell operation), 18, 422
- ACCLAIM project, xl
- accumulator, 142–144
 - breadth-first traversal, 159
 - declarative state, 416
 - depth-first traversal, 158
 - for** loop, 192
 - limit of declarative model, 315
 - loop abstraction, 262
 - loops, 187
 - parser, 163
 - Prolog, 143
 - relation to difference lists, 144
 - relation to fold operation, 187
 - tree traversal, 195
- ACID properties, 609
- action (in GUI), 693
- active memory, 75
 - size, 175, 314
- ActiveX, 466
- adder
 - n -bit, 347
 - full, 273
- address
 - IP, 209
 - URL, 214
- adjacency list, 468
- Adjoin** operation, 440, 828
- AdjoinAt** operation, 440, 828
- adjunction (environment), 63
- ADT (abstract data type), *see* type
- adversary, 211
- agent, 357
 - concurrent component, 370
 - message-passing approach, 584
- Alarm** operation, 309, 401
- alarm clock, xxix
- algebra, xxxvii, 115
- algorithm
 - breadth-first traversal, 159
 - cached state, 739
 - compression, 180
 - copying dual-space garbage collection, 79
 - Dekker's algorithm for mutual exclusion, 578
 - depth-first traversal, 158
 - distributed garbage collection, 740
 - distributed locking, 727
 - distributed unification, 739
 - elevator, 384
 - Flavius Josephus problem, 565
 - Floyd-Warshall, 473
 - garbage collection, 77
 - Hamming problem, 298
 - linear congruential generator, 478
 - mergesort, 140, 169
 - generic version, 184
 - mobile state, 739
 - Newton's method for square roots, 122
 - nonalgorithmic programming, 634
 - parallel transitive closure, 473
 - Pascal's triangle, 12
 - quicksort, 235, 531
 - random number generation, 476
 - termination detection, 390
 - thread scheduling, 245
 - transitive closure, 467
 - tree, 154
 - tree drawing, 161, 278
 - unification, 102
 - word frequency, 201
- aliasing, 426, 452
- allocation, 76
- always true (in temporal logic), 611
- Anderson, Ross J., 750, 845
- Andrews, Gregory, 590
- andthen** operator, 84
- Append** operation, 831
- apple, 493
- Apple Corporation, xl
- applet, 724
- application
 - FlexClock example, 707
 - ping-pong example, 310
 - standalone, 225
 - Java, 561
 - video display, 321

- applicative order reduction, 336
- Aquarius Prolog, 143
- arbitrary precision integer arithmetic, 6, 823
- arch, xxxii
- arithmetic, 56
- Arity** operation, 56, 828
- arity, 442, 828
- Armstrong, Joe, 590
- Arnold, Ken, 543
- array, 439
 - extensible, 443
 - usage trade-offs, 442
- ASCII (American Standard Code for Information Interchange), 462, 721
- ask operation, 786, 793
- assert/1** operation (in Prolog), 669, 675
- assertion, 448
- assignment
 - axiomatic semantics, 449
 - cell operation, 18, 422
 - in-place, 314
 - Java, 559
 - monotonic, 851
 - multiple, 851
 - single, 44, 851
 - to iteration variable, 191
- association, 537, 850
- association list, 491
- asynchronous failure detection, 745
- ATM (Asynchronous Transfer Mode), 395
- atom, 53, 825
 - defining scope, 514
 - predicate calculus, 645
 - propositional logic, 644
- atomic action, 588–625
 - approaches to concurrency, 581
 - reasoning, 588
 - when to use, 584
- atomicity, 23, 609
- AtomToString** operation, 826
- attribute
 - final (in Java), 550, 558
 - initialization, 503
 - object, 502
- availability, 717
- AXD301 ATM switch, 395
- axiom, 646
- axiomatic semantics, 40, 444–453, 644
- Backus, John, 34
- bagof/3** operation (in Prolog), 638, 683
- Bal, Henri, 340
- batching
 - distributed stream, 725, 730
 - object invocations, 546, 572
- Baum, L. Frank, 3
- Bernstein, Philip A., 609
- billion dollar budget, 527
- binary integer, 822
- binding
 - basic and nonbasic, 792
 - dynamic, 511
 - static, 512
 - variable-variable, 48
- bistable circuit (digital logic), 274
- blank space (in Oz), 843
- Blasband, Darius, 654
- block
 - file, 297
 - imperative language, 491
 - memory, 76, 77
 - Smalltalk, 549
- blocking operation, 243
- blue cut (in Prolog), 679, 681
- book
 - Component Software: Beyond Object-Oriented Programming*, 466
 - Concurrent Programming in Erlang*, 590
 - Concurrent Programming in Java*, 589
 - Concurrent Programming: Principles and Practice*, 590
 - Object-Oriented Software Construction*, 495
 - Software Fundamentals*, 466
 - Structure and Interpretation of Computer Programs*, xxxiii
 - The Art of Prolog*, 679
 - The Mythical Man-Month*, 466
 - Transaction Processing: Concepts and Techniques*, 590
- boolean, 53

- Borges, Jorge Luis, 633
- bottom-up software development, 455
- bound identifier occurrence, 65
- bounded buffer, 268
 - data-driven concurrent version, 268
 - lazy version, 295
 - monitor version, 602
- brand, 504
- Brand, Per, xl
- breadth-first traversal, 159
- break** statement, 491
- break operation, 491
- bridge building, xxx, xxxi
- Brinch Hansen, Per, 600
- Brooks, Jr., Frederick P., 466
- Browser tool, *see* Mozart Programming System
- by-need execution, 286–289, 798
 - waitQuiet** operation, 800
 - Multilisp, 342
 - Oz 3, 811
- ByNeed** operation, 287
- byte (in Java), 560
- bytestring, 833
- calculator, 3
- calculus
 - analysis, 173
 - first-order predicate, 645
 - foundational, xxxi
 - λ , xxxi, 43, 98, 336, 351, 795, 807, 813, 848
 - π , xxxi, 43, 55, 807
- call/1** operation (in Prolog), 675
- call by ..., *see* parameter passing
- call graph, 318
- capability, 211
 - declarative, 213
 - method label, 516
 - revocable, 433, 488
- Cardelli, Luca, 728
- cascading abort, 613
- case** statement, 68, 793
- catch** clause (in **try**), 95
- causality, 241, 361
- Ceil** operation, 824
- cell (explicit state), 417, 422–425, 797, 850
- cellular phone, xxix, xxxii
- channel
 - asynchronous, 355, 393
 - component interface, 459
 - dataflow variable, 337
 - many-shot, 371
 - one-shot, 371, 376
 - port, 354
 - synchronous, 629
- character, 822
 - alphanumeric, 51, 843
 - Java, 560
- choice** statement, 636, 778
- chunk, 829
 - reflection, 524
- Church-Rosser Theorem, 336
- Churchill, Winston, 453
- Clarke, Arthur C., 106, 314
- class, 421, 493, 495, 497, 501, 552
 - abstract, 528, 552
 - active object, 564
 - common limitations, 545
 - complete ADT definition, 498
 - concrete, 529, 532, 552
 - delegation, 517
 - diagram, 534, 539
 - encapsulation control, 512
 - event manager example, 570
 - forwarding, 517
 - generic, 531
 - higher-order programming, 532
 - implementation, 554
 - incremental ADT definition, 507
 - inheritance, 508
 - inner, 546
 - inner (in Java), 558
 - introduction, 20
 - member, 502
 - metaclass, 553
 - mixin, 572
 - parameterized, 531
 - patching, 527
 - programming techniques, 524
 - reflection of object state, 523
 - structure view, 525

- substitution property, 525, 527
- type view, 525
- clause
 - case** statement, 84
 - definite, 662
 - Erlang, 396
 - Horn, xxxv, 647, 652, 662, 675
- clock
 - context-sensitive, 707
 - digital circuit, 275
 - digital logic, 271
 - synchronization, 313
- clone
 - array, 440
 - computation space, 774
 - dictionary, 441
 - object, 523
- closed distribution, 737
- closure, *see* procedure value
- cluster, 717
- code duplication, 496
- code generator, 164
- coherence
 - between graphical views, 704
 - cache, 463, 739
 - network transparency, 726
- collector, 192, 326, 434, 488
- COM (Component Object Model), 466
- combinations, 6
- combinator, 282
- comment (in Oz), 843
- common self, 517
- compaction, 77
- comparison, 57
- compilation
 - separate, 107
 - standalone, 232
 - ping-pong example, 310
 - unit, 225, 420, 457, 818
- compiler, 163, 674
 - extensible, 548
 - Prolog, 676
 - smart-aleck, 314
- Compiler Panel tool, *see* Mozart Programming System
- complete value, 47
- completeness
 - Turing, 848
- complexity
 - amortized, 149, 177, 440
 - asymptotic, 169
 - banker's method, 178
 - big-oh notation, 170
 - introduction, 12–13
 - physicist's method, 178
 - space, 175
 - time, 13, 169
 - worst-case, 169, 175
- component, 114, 185, 223
 - abstraction, 462
 - avoiding dependencies, 463
 - diagram, 360
 - encapsulates a design decision, 462
 - functor, 420
 - further reading, 466
 - future role, 464
 - graph, 465
 - implementation, 224
 - interface, 224
 - module, 420
 - software, 224
- compositionality, 419, 631
 - class, 507
 - encapsulated search, 638
 - exception handling, 92
 - Solve** operation, 638
- computation, 63
 - iterative, 120
 - recursive, 126
 - secure, 210
- computation model, xxvii, 31
 - active object*, 563
 - constraint-based*, 639, 653, 764
 - data-driven concurrent*, 240
 - declarative concurrent with exceptions*, 332
 - declarative concurrent*, 239
 - declarative with exceptions*, 93
 - declarative*, 50
 - demand-driven concurrent with exceptions*, 333
 - demand-driven concurrent*, 240, 286
 - descriptive declarative*, 117
 - distributed*, 718

- graph notation, 741
- extended relational*, 673
- general*, 845
- job-based concurrent*, 628
- lazy concurrent*, 240, 286
- maximally concurrent*, 585, 628, 772
- message-passing concurrent*, 354
- nondeterministic concurrent*, 402, 653
- object-based*, 543
- object-oriented*, 496
- order-determining concurrent*, 277
- relational*, 635
- secure declarative*, 205
- shared-state concurrent*, 581
- stateful concurrent*, 584, 786
- stateful*, 421
- strict functional*, 99
- atomic action, 588
- closed, 326
- concurrent constraint, 810
- concurrent logic programming, 292, 402, 813
- design principles, xxxiii, 845
- determinate concurrent constraint programming, 343
- determinate parallel programming, 343
- deterministic logic programming, 648
- Erlang, 394
- Haskell, 328
- insecure, 326
- Java, 557, 625
- nondeterministic logic programming, 650
- open, 326
- partial failure, 326, 744
- Prolog, 674
- secure, 326
- semantics, 783
- using models together, xxxiv, 325
- computation space, 666, 759
- computer engineering, xxxvi
- computer science, xxxvi
- computing, *see* informatics
- conc** statement, 283
- concurrency, 322
 - competitive, 259
 - cooperative, 259
 - dataflow, 17
 - declarative, 237, 247, 806
 - Erlang, 394
 - further reading, 589
 - importance for programming, xxx
 - interactive interface, 90
 - interleaving, 22
 - introduction, 16
 - Java, 625
 - monitor safety, 601
 - nondeterminism, 21
 - order-determining, 277
 - practical approaches, 581
 - queue, 592
 - rules of thumb, 584
 - teaching, xxxvi
 - transaction, 610
- concurrency control, 573, 611, 751
- concurrent composition, 281, 393
- condition variable, 607
- conditional critical section, 605
- configuration, 787
- confinement
 - failure in declarative model, 250
 - partial failure, 715
 - transaction, 610
- confluence, 336, 807
- connector, 326
- cons cell (list pair), 7, 54, 830
- consistency (in transaction), 609
- consistency protocol, 718
- constraint, 248
 - tree drawing, 161
- constructor, 560
- consumer, 262
- context, 92
- context-sensitive
 - grammar, 36
 - GUI design, 707
- continuation, 385
 - procedure, 367
 - record, 366
- contract, 418
- contrapositive law, 644
- control abstraction
 - try – finally**, 349

- break, 491
- concurrent composition, 282
- CSP communication, 629
- Erlang mailbox, 410
- iteration, 125
- loops, 186
- need for higher-order, 182
- Smalltalk block, 549
- coordination model, 460, 594
- CORBA (Common Object Request Broker Architecture), 364, 466
- coroutine, 279, 459
 - relation to laziness, 290, 293, 582
- correctness, 644
 - introduction, 11–12
- Cray-1 computer, 178
- critical region, 590
- cryptography
 - unique name generation, 210
- curriculum (informatics), xxxvi
- currying, xxxv, 196–197, 236
 - Haskell, 329
- cyclic structures, 104
- dangling reference, 76, 183, 463
- Danvy, Olivier, 236
- data structure
 - active, 78
 - class, 501
 - cyclic, 102
 - dictionary, 199
 - difference list, 144
 - difference structure, 144
 - ephemeral, 150
 - external, 79
 - graph, 468
 - higher order, 185
 - infinite, 14
 - list, 54, 132
 - long-lived, 80
 - partial, 47
 - persistent, 303
 - protected, 206, 429
 - queue, 149
 - record, 53
 - recursive, 133
 - size, 176
 - stack, 198
 - store, 76
 - tree, 153
 - tuple, 53
- data type, *see* type
- data-driven execution, *see* eager execution
- database, 667
 - deductive, 668
 - distributed, 625
 - in-memory, 176
 - Mnesia (in Erlang), 395
 - persistence, 667
 - query, 668
 - relational, 667
 - shared-state concurrency, 584
 - transaction, 608
 - update, 668
- dataflow, 61
 - channel, 337
 - declarative model, 18
 - error, 91
 - examples, 252
 - I-structure, 342
 - introduction, 17
 - lazy execution, 290
 - parallel transitive closure, 473
 - rubber band, 255
 - variable, 44, 49, 50
- Date, C.J., 668
- Dawkins, Richard, 418
- DBMS (Database Management System), 668
- DCOM (Distributed Component Object Model), 466
- deadlock, 613
 - callback, 365
 - concurrency control, 613
 - detection, 614
 - digital logic simulation, 274
 - lazy execution, 293
 - prevention, 614
- deallocation, 76
- debugging, 522
 - dangling reference, 77, 183
 - declarative component, 314
 - distributed application, 751

- erroneous suspension, 50, 91
- inspecting object attributes, 507
- declarative, 113, 415
- declarative concurrency, *see* concurrency, declarative
- declarative program, 248
- declare** statement, 4, 88
 - syntax convention of book, xliii, 88
- define** clause (in functor), 224
- Definite Clause Grammar (DCG), 143, 662
- Delay** operation, 309
- delay gate (digital logic), 274
- delay** operation (in Multilisp), 342
- delay point, 588
- delegation, 518–522
 - otherwise** method, 506
- Dell Corporation, xl, 204, 475
- demand-driven execution, *see* lazy execution
- De Morgan's law, 644
- denotational semantics, 40
- dependency
 - component, 227
 - dataflow, 341
 - declarative component, 314, 324
 - grammar context, 36
 - inheritance, 421
 - order-determining concurrency, 277
 - removal of sequential, 393
 - sequential model, 419
 - word frequency application, 231
- depth-first traversal, 158
- dereferencing, 47
- design methodology
 - concurrent program, 372
 - language, 42, 335, 549, 813, 852
 - large program, 454
 - small program, 221
- design patterns, xxxv, 421, 493, 540–543
 - Composite pattern, 540
- destructive assignment
 - interaction with call by name, 490
- determinism (declarative programming), 113
- diagnostics, 633
- dialog model, 704
- dictionary, 199, 440
 - declarative, 199
 - efficiency, 203
 - internal structure, 203
 - list-based implementation, 200
 - relation implementation, 671
 - relation to record, 441
 - relation to stream, 442
 - secure declarative, 210
 - standalone, 228
 - stateful, 201
 - tree-based implementation, 200
 - tuple space implementation, 596
 - usage trade-offs, 442
 - word frequency implementation, 475
 - Word of Mouth simulation, 482
- difference list, 144
- difference structure, 144
- digital logic, 271
 - satisfiability problem, 179
- Dijkstra, Edsger Wybe, 444
- directed graph, 467
- disconnected operation, 747
- discriminant, 181
- disentailment, 105–106, 785
- distributed lexical scoping, 728
- distributed system, 326, 358, 363, 392, 713
 - closed, 737
 - open, 514, 715, 717, 720
- Distribution Panel tool, *see* Mozart Programming System
- div** (integer division) operation, 56, 823
- divide-and-conquer, 140
- domain model, 703
- domain of discourse, 645
- dotted pair, 7, 830
- Duchier, Denys, xl, 389
- durability, 609
- dynamic
 - binding, 511
 - linking, 225, 289
 - scope, *see* scope, dynamic
 - typing, 106–108
- Dynamic HTML, 692

- eager execution
 - declarative model, 99
 - distributed producer/consumer, 725
 - introduction, 13
 - producer/consumer stream, 261
 - relation to synchronization, 339
 - strictness, 336
- eager failure detection, 745
- EBNF (Extended Backus-Naur Form), 34, 662, 663, 685
- Einstein, Albert, 283
- Elcock, E. W., 634
- elephant, xxvii
- elevator, *see* lift
- Emacs text editor, 817
- embedding, 185
- encapsulated
 - search, 637
 - state, introduction, 18
- encapsulation, 419
- Encyclopaedia Britannica (11th edition), 493
- Ende, Michael, 353
- entailment, 100, 105–106, 785
- Enterprise Java Beans, 465
- environment, 46, 62
 - adjunction, 63
 - calculating with, 63
 - contextual, 67
 - interactive interface, 88
 - restriction, 64
- ephemeral data structure, 150, 303
- equality
 - structure, 426, 729
 - token, 426, 720, 729
- Ericsson (Telefonaktiebolaget LM Ericsson), 327, 394
- Erlang, 394–402
- error
 - absolute, 122
 - domain, 97
 - relative, 122
 - type incorrect, 97
 - variable not introduced, 836
 - variable used before binding, 49
- error logging, 570
- Escher, Maurits Cornelis, 689
- Euclid, 3
- event (in GUI), 693
- event handler, 569
- event manager
 - active objects, 569
 - adding functionality with inheritance, 571
- eventually true (in temporal logic), 611
- evolutionary software development, 455
- example programs (how to run), xliii
- exception, 91–97, 549, 849
 - distribution fault, 749
 - error**, 97, 803
 - failure**, 97, 803
 - system**, 97
 - uncaught, 95, 803
- Exchange** operation, 424
- exchange operation
 - on cells, 424
 - on object attributes, 502
- exclusive-or operation, 16
- execution state, 62
- exercise
 - (*advanced exercise*), xxxviii
 - (*research project*), xxxviii
- explicit state, *see* state
- Explorer tool, *see* Mozart Programming System
- Explorer tool (in Mozart), 763
- export** clause (in functor), 224
- expression, 82
 - andthen** operator, 84
 - basic operation, 56
 - basic value, 51
 - ill-formed, 93
 - λ notation, 98, 330
 - nesting marker (\$), 85
 - normal order reduction, 336
 - orelse** operator, 84
 - procedure value, 66
 - receive** (Erlang), 399
- external resource, 79
- extreme programming, 456
- factorial function, 4–25, 127–130, 234, 626
- fail** statement, 636

- FailedValue** operation, 334
- failure
 - declarative concurrency, 249
 - detection, 745
 - exception, 97
 - failed value, 334
 - inheritance, 496
 - negation as failure, 675
 - partial, 326
 - software project, 527, 655
 - transaction, 609
 - unification, 103
- fairness
 - stream merger, 406
 - thread scheduling, 243, 245, 256, 339
- false**, 826
- fault model, 745
- fault tolerance
 - Erlang, 395
- FCFS (First-Come-First-Served), 375
- feature (record field name), 826
- FGCS (Fifth Generation Computer System), Japanese project, 405
- FIFO (First-In-First-Out), 149, 235, 343, 355, 375, 387, 459, 725, 730
 - Erlang, 394
 - message sending, 364
 - port, 355
- file, 213
- Filter** operation, 194, 831
- finalization, 79, 486
 - lazy execution, 486
- finally** clause (in **try**), 95
- finite failure, 675
- finite state machine
 - event manager, 570
- firewall, 326
- first-argument indexing, 669
- fixpoint, 772
- Flatten** operation, 145, 235, 301
- Flavius Josephus problem, 565–567
- floating point
 - arithmetic, 823
 - number, 822
- FloatToInt** operation, 824
- Floor** operation, 824
- flow control, 100, 266
 - lazy execution, 266
 - thread priority, 269
- Floyd, Robert W., 444, 634
- fold operation, 187
 - FoldL**, 188, 469, 831
 - FoldR**, 188
 - stream objects, 263
- for** statement, 190
- for** loop, 451
- ForAll** operation, 831
- formal language, 35
- forwarding, 518
- foundational calculus, xxxi
- framework
 - computation model as, 846
- France, 419
- free identifier occurrence, 59, 65, 645
- free memory, 76
- freeze/2** operation (in Prolog), 675
- fresh
 - name, 206
 - variable, 790
- full adder, 273
- fun** statement, 85
- function, 85
 - incremental, 302
 - introduction, 4
 - lazy, 800
 - monolithic, 302
 - monotonic, 851
 - partially-applied, 197
 - state transition, 359
- functional decomposition, 138, 547
- functional look, 198
- functor, 224, 457, 715
 - interactive use, 818
 - main, 225
 - Oz 3, 811
- functor** statement, 227
- future, 342
- future** operation (in Multilisp), 342
- Gamma, Erich, 540
- garbage collection, 75, 77
 - copying dual-space, 79
 - distributed, 740
 - external references, 485

- generational, 80
- lease-based, 740, 744
- real-time, 78
- root set, 77
- weighted reference counting, 740
- gate** statement, 277
- gate (digital logic), 271
- Gelernter, David, 594
- generalization, 575
- generate-and-test, 641
- generating function, 173
- genericity, 183–185
 - object-oriented programming, 531
 - static and dynamic, 532
- global condition, 158
- glue (in GUI), 698
- Glynn, Kevin, 327
- Gödel's Incompleteness Theorem, 646
- Gosling, James, 543
- grammar, 34–38
 - ambiguous, 36, 169
 - context-free, 35
 - context-sensitive, 36
 - definite clause (DCG), 143, 662
 - disambiguation, 36
 - EBNF (Extended Backus-Naur Form), 34, 685
 - left-recursive, 656
 - natural language, 655
 - nonterminal symbol, 34, 165
 - terminal symbol, 34, 165
 - unification, 662
- graph
 - bushy, 465
 - component, 465
 - distribution model, 741
 - Haskell expression, 328
 - hierarchical, 465
 - implementation, 468
 - nonlocal, 465
 - Small World, 465
- Gray, Jim, 590, 609
- green cut (in Prolog), 682
- grue cut (in Prolog), 682
- guarded method, 605
- guardian, 486
- GUI (Graphical User Interface)
 - AWT (Abstract Window Toolkit) package, 689
 - component-based programming, 464
 - design, 689
 - Haskell fudgets, 689
 - model-based, 703
 - Qt module, 690
 - read-only view, 704
 - Swing components, xxxv, 689
 - text input/output, 216
 - Visual Studio, 689
- Halting Problem, 212, 691
- Hamming problem, 298, 349
- Hamming, Richard, 298
- handler
 - event, 569
 - exception, 91, 92
 - finalization, 486
 - GUI design, 216, 693, 692–698
- HasFeature** operation, 828
- hash table, 442
- Haskell, 327–331
- Helm, Richard, 540
- Herbert, Frank, 755
- hexadecimal integer, 822
- Hoare, Charles Antony Richard, 235, 444, 600
- Horn clause, xxxv, 647, 652, 662, 675
- HTML (HyperText Markup Language), 117, 689
- Hudak, Paul, 98, 314
- Hughes, John, 284
- IBM Corporation, 43
- IDE (Interactive Development Environment), xliii, 817
- identifier, 4, 46
 - bound occurrence, 65
 - escaped, 515
 - free occurrence, 59, 65, 645
- IEEE floating point standard, 550, 822
- if** statement, 68, 793
- ILOG Solver, 676
- impedance matching, 325
 - concurrency, 585
 - event manager example, 574

- imperative, 415
- implementation, 418
- import** clause (in functor), 224
- inactive memory, 76
- incompleteness, 646
- inconsistency, 791
- incremental function, 300
- independence (declarative programming), 113
- infinite precision arithmetic, 6
- infinity
 - floating point, 824
- informatics, *xxxvi*
 - curriculum, *xxxvi*
 - success of object-oriented programming, 493
 - usefulness of computation models, *xxviii*
- Information Cities project, 421
- information hiding, 493
- information systems, *xxxvi*
- inheritance, 21, 421, 493, 495, 508–543
 - avoiding method conflicts, 516
 - cautionary tale, 526
 - design patterns, 540
 - directed and acyclic, 509
 - event manager, 571
 - factoring, 496
 - generalization, 575
 - implementation, 556
 - implementation-sharing problem, 539
 - Java, 558
 - multiple, 508, 533
 - rules of thumb, 539
 - simple, 508
 - single, 508, 537
 - software reuse, 496
 - static and dynamic binding, 511
 - structure view, 525
 - syntactic aspect, 496
 - thread priority, 258
 - type view, 525
 - upwards, 575
- inherited argument, 163
- instantiation, 185, 419
- integer
 - tilde ~ as minus sign, 822
- interactive system
 - importance of worst-case complexity, 177
- interface
 - general concept, 224
 - Java, 558
 - Runnable** (in Java), 625
- interface builder, 689
- interleaving semantics, 241, 784
- Internet, 209, 361, 713, 717, 723
 - simulation of Web use, 481
- interoperability, 108
- interpreter, 43
 - approach to define semantics, 43
 - generic parser, 662
 - GUI description, 712
 - meta, 667
 - metacircular, 43
 - original Erlang, 396
- IntToFloat** operation, 824
- invariant, 137, 390, 420, 444
- IsAtom** operation, 826
- IsChar** operation, 824
- IsDet** operation, 320, 321, 338, 402, 673, 850
- IsLock** operation, 591
- ISO 8859-1 character code, 822, 825
- isolation, 609
- IsProcedure** function, 57
- IsRecord** operation, 828
- I-structure, 342, 473, 814
- IsTuple** operation, 829

- Janson, Sverker, xl, 675
- Java, 556–563, 625–626
- JavaBeans, 466
- Jefferson, Thomas, 11
- Johnson, Ralph, 540
- journaling, 538

- Kahn, Gilles, 343
- kernel language, *see* computation model
- kernel language approach, *xxx*, 38–43
 - choice of formalism, *xxxiii*
- keywords (table of), 841
- knowledge representation, 850
- Knuth, Donald Erwin, 173, 477

- Kowalski, Robert, 415
 KWIC (keyword in context) index, 679
- Label** operation, 56, 828
 label (record identification), 20, 53
 λ calculus, xxxi, 43, 98, 336, 351, 795, 807, 813, 848
 LAN (local-area network), 361, 723, 745
 language
 AKL, xl, 674, 810
 Absys, 415, 634
 Ada, 436, 629
 Algol, 415, 493
 declarative concurrent, 343
 nondeterministic, 634
 Alice, 108
 C++, 45, 49, 76, 183, 340, 449, 491, 493, 497, 510, 514, 516, 541, 546, 551, 557
 C-Linda, 594
 CLOS (Common Lisp Object System), 522
 CSP (Communicating Sequential Processes), 629
 Clean, 331
 Cobol, 551, 654
 parsing problem, 654
 Common Lisp, 60, 193
 Concurrent Haskell, xxxiv
 Concurrent ML, xxxiv, 852
 Concurrent Prolog, 405
 C, 76, 182
 Eiffel, 525, 852
 Erlang, 77, 99, 327, 394–402, 460, 551, 570, 590, 809, 852
 E, 211
 FCP (Flat Concurrent Prolog), 405, 809
 FP, 335
 Flat GHC, 405
 Fortran, 415, 654
 parsing problem, 654
 GHC (Guarded Horn Clauses), 405
 Haskell, xxviii, xxx, 45, 77, 99, 118, 140, 197, 277, 284, 291, 327–331, 335, 336, 340, 343, 349, 461, 551, 689, 809
 IC-Prolog, 405
 Id, 343, 814
 Java, xxviii, xxx, 43, 45, 49, 77, 183, 340, 434, 449, 466, 491, 493, 497, 510, 514, 516, 541, 546, 549–551, 556–563, 574, 589, 600, 625–626, 689, 809, 852
 monitor, 601
 Leda, xxxi
 Linda extension, 594, 629, 810
 Lisp, xxviii, 7, 60, 77, 132, 415, 663, 830
 ML, *see Standard ML, Concurrent ML, Objective Caml*
 Mercury, xxviii, 118, 331, 675, 809
 Miranda, 284, 349
 Multilisp, 342, 814
 Objective Caml, 549
 Obliq, 728
 Oz 1, Oz 2, Oz 3, 810–811
 Oz, xxxiii, xl, 513, 551, 676, 809, 846, 852
 Parlog, 405
 Pascal, 163, 182, 434, 809
 Prolog, xxviii, xxx, xxxv, 7, 32, 49, 77, 118, 143, 145, 277, 292, 335, 340, 396, 405, 415, 551, 634, 647, 652, 654, 662, 667, 669–673, 673–684, 809, 852
 pure, 652
 SICStus, 193, 676
 Scheme, xxviii, xxx, xxxi, xxxiv, 32, 45, 60, 99, 291, 551, 809
 Simula, 415, 493
 Smalltalk, xxviii, 45, 77, 340, 493, 497, 513, 514, 516, 522, 546, 549, 550, 852
 Standard ML, xxviii, xxxiv, 32, 45, 99, 118, 140, 197, 291, 331, 335, 809
 Visual Basic, 465
 pH (parallel Haskell), 343, 814
 tcl/tk, 689, 690, 712
 assembly, 212, 314, 415, 557, 634
 coordination, 594
 formal, 35
 multiparadigm, xxxi

- non-strict, 336
 - popular computation models, 809
 - practical, 33
 - secure, 211
 - specification, 119
 - symbolic, 55, 551
- language design
 - abstraction lifecycle, 42
 - declarative, 335
 - golden age, 415
 - layered, 852
 - lazy execution, 335
 - object properties, 549
 - trade-offs, 813
- Lao-tzu, 283
- last call optimization, 74
- latch (digital logic), 275
- late error detection (at run-time), 509
- latency, 268
 - tolerance, 341
- LaTeX document, 463
- Latin-1, 462
- layered language design, 852
- lazy evaluation, 99
 - coroutining, 582
 - example, 13
 - explicit, 185
 - Haskell, 328
 - relation to call by need, 438, 490
 - relation to non-strict evaluation, 336
 - schedule, 350
 - strictness analysis, 295, 329, 349
- lazy execution, 283
 - bounded buffer, 267
 - flow control, 266
 - Hamming problem, 298
 - higher-order programming, 196
 - incremental, 300
 - introduction, 13
 - monolithic, 302, 348
 - needs finalization, 486
 - relation to synchronization, 340
- lazy failure detection, 745
- Lea, Doug, 589
- legal program, 33
- Length** operation, 831
- lex/yacc parsing, 654
- lexical analyzer, 34
- lexical scope, *see* scope, lexical
- lexical syntax (of Oz), 843
- lexically-scoped closure, *see* procedure value
- lexicographic order (of atoms), 57, 825
- Ley, Willy, 633
- library, 232
- lifecycle
 - abstraction, 42
 - memory block, 76
- lift control system, 374
- lifting
 - booleans to streams, 277
 - serializability, 609
 - synchronization, 366
- lightweight transaction, 609
- Linda (tuple space), 594
- linguistic abstraction, *see* abstraction, linguistic
- linking, 214, 225, 227, 232, 458, 819
 - component, 226, 462
 - dynamic, 289
 - failure detection in Erlang, 395
- Linux operating system, xl, 204, 475, 504
- list, 54, 131, 830
 - difference, 144
 - advantage, 149
 - flattening, 145
 - introduction, 6
 - nested, 138
 - usage trade-offs, 442
- list pair, 54, 830
- literal, 825
- liveness, 611
- Llull, Ramón, 633
- local** statement, 58, 64, 790
- lock, 588, 590–591
 - get-release, 606
 - implementation, 599
 - introduction, 23
 - Java, 626
 - read, 630
 - simple, 599
 - thread-reentrant, 600
 - transaction, 611

- write, 630
- lock** statement, 23, 591
- locus of control, 279
- logic
 - gate, 271
 - predicate calculus, 645
 - propositional, 644
 - temporal, 611
- logic programming, 644, 646
 - process model, 403, 809
- logical equivalence, 247, 788
 - configuration, 806
- logical formula, 645
- logical semantics, 40, 644–653
- logical sentence, 645
 - assertion, 445
 - invariant, 444
- Louis XIV, 413, 418
- Lynch, Nancy, 361
- Mac OS X operating system, xxxviii, xl, 258
- Macintosh computer, xl
- MacQueen, David, 343
- Maher, Michael, 675, 810
- mailbox, 460
 - Erlang, 394, 397
 - implementation, 398
- maintenance, 461
- MakeRecord** operation, 828
- MakeTuple** operation, 381, 829
- Manchester Mark I, 38
- Manna, Zohar, 444
- many-to-one communication, 358
- Map** operation, 193, 470, 831
- mathematical induction, 11
- matrix
 - graph representation, 468
 - list of lists implementation, 235
- Max** operation, 196
- measurement accuracy
 - constraint programming, 757
- Member** operation, 831
- memoization, 425, 460
 - call by need, 438
 - declarative programming, 315
 - explicit state, 27, 703
 - GUI example, 703
 - unification, 104
- memory
 - address in abstract machine, 57
 - consumption, 175
 - content-addressable, 595
 - leak, 76, 77
 - Prolog, 681
 - lifecycle, 76
- memory management, *see* garbage collection
- message, 504
- message-passing concurrency, *see* object, active, *see* object, port
- meta-interpreter, 667, 685
- meta-object protocol, *see* protocol, meta-object
- method
 - object, 21, 502
 - wrapping, 522
- methodology, *see* software development
- Meyer, Bertrand, 453, 495, 533
- Microsoft Corporation, 466, 689
- middle-out software development, 455
- mind of programmer
 - capabilities (atoms vs. names), 516
 - difference list, 149
 - language design trade-offs, 813
 - order-determining concurrency, 277
 - state (implicit vs. explicit), 416
 - use of constraints, 279
- minus sign (use of tilde ~), 822
- Mnesia (Erlang database), 395
- mod** (integer modulo) operation, 56, 823
- model
 - logic, 645
- model-based GUI design, 703
- modularity, xxxv, 315, 417, 462
 - encapsulated search, 638
 - reengineering, 527
- relation to concurrency, 243, 319
- relational model, 673
- system decomposition, 460
- module, 185, 224, 457
 - Array**, 439
 - Atom**, 826
 - Browser**, 228

- Char, 823, 824
- Compiler, 817
- Connection, 721, 737
- Dictionary, 440
- Distribution (supplement), 724
- Fault, 745, 749
- File (supplement), 214, 298, 571
- Finalize, 485
- Float, 823
- Int, 823
- List, 263, 393, 831
- Module, 227, 421, 459, 736, 818
- MyList (example), 225
- Number, 16, 56, 185, 823
- OS, 377, 618, 701, 708, 736
- ObjectSupport, 523
- Open, 571, 735
- Pickle, 219
- Property, 95, 258, 260
- Qt, 216, 690, 735
- Record, 828
- Remote, 260, 737
- Space, 666
- String, 826
- Thread, 260, 280
- Time, 309
- Tk, 712
- Tuple, 829
- Value, 334, 832
- Base, 226, 233
- compilation unit, 457
- dynamic linking, 290
 - failure, 334
- dynamic typing, 107
- importing, 227
- interface, 459
- library, 232
- resource, 735, 752
- specification, 224
- System, 226, 233
- MOGUL (Mozart Global User Library), 225
- monad, xxxv, 327, 337
- monitor, 588, 600–608
 - condition variable, 607
 - guarded method, 605
 - implementation, 605
- Java language, 625
- Java semantics, 601
- monolithic function, 302
- monotonicity, 851
 - constraint programming, 772
 - dataflow variable, 341, 578
 - need predicate, 799
 - need property, 288
 - store, 785
 - thread reduction, 243, 785, 786
- Moore's Law, 178
- Morrison, J. Paul, 261
- Mozart Consortium, xxxviii, xl, 108
- Mozart Programming System, 259
 - Browser tool, 4, 89
 - displaying cyclic structures, 104
 - lists, 830
 - WaitQuiet operation, 800
- cheap concurrency, 256
- Compiler Panel tool, 817
- Distribution Panel tool, 817
- Explorer tool, 817
- garbage collection, 79
- interactive interface, 3, 88, 817
- kernel languages, 846
- library modules, 233
- limited role of the compiler, 510
- Open Source license, xxxviii
- overview, xxxviii
- Panel tool, 817
- performance, 204, 387
- separate compilation, 461
- thread scheduler, 257
- uncaught exception, 95, 803
- multi-agent systems (MAS), xxxvii, 370, 584
- multimedia, 179
- multiprocessor
 - cluster, 717
 - shared-memory, 716
- mutable store (for cells), 797
- Myriorama, 277
- name, 206, 720, 795, 825, 849
 - defining scope, 514
 - distributed computing, 717
 - fresh, 206

- generation, 209
- name server, 411, 718
- Naur, Peter, 34
- needed variable, 288, 799
- negation as failure, 675
- nesting marker, 54, 85, 363, 373
- network awareness, 395, 729
- network transparency, 260, 395, 714
- neutral element, 188
- New** operation, 501, 555
- NewActive** operation, 564
- NewActiveExc** operation, 569, 734
- NewArray** operation, 439
- NewCell** operation, 424
- NewDictionary** operation, 440
- NewLock** operation, 23, 591
- newName** operation, 206, 795, 825
- NewPort** operation, 356
- NewStat** operation, 731
 - resilient, 748
- Newton's method for square roots, 122
- Newton, Isaac, 283
- nil**, 830
- noise (electronic), 477
- non-strict evaluation, 336
 - Haskell, 328
- nonblocking operation, 338
 - receive** (in Erlang), 402
 - receive, 338
 - send, 338
 - stack, 585
- noncompositional design, 465
- nondeterminism
 - choice** statement, 633, 635, 778
 - declarative concurrent model, 583
 - don't know, 325, 634
 - introduction, 21
 - limitation of declarative model, 319
 - logic programming, 650
 - observable, 22, 238, 319, 578, 581, 583, 584
 - bounded buffer, 603
 - Filter** operation, 394
 - lack of in Flavius Josephus problem, 567
 - relation to exceptions, 332
 - relation to coroutines, 280
 - thread scheduler, 257
- nonvar** operation (in Prolog), 673, 675
- normal order reduction, 336
- notify** operation (in monitor), 600
- notifyAll** operation (in monitor), 601
- NP-complete problems, 179
- number, 53, 821
- O'Keefe, Richard, 113, 416, 681
- object, 421, 497, 549, 552
 - declarative, 431, 488, 575
 - introduction, 19
 - strong, 549
- object code, 225
- object graph, 560
- object, active, 357, 563
 - defining behavior with a class, 564
 - example, 564
- object, port, 358, 421
 - approaches to concurrency, 581
 - Erlang, 570
 - further reading, 590
 - many-to-one communication, 358
 - reactive, 359
 - reasoning, 360
 - sharing one thread, 385
 - when to use, 584
- object, stream, 261, 270–271, 419, 421, 582
 - comparison with port object, 358
 - Flavius Josephus problem, 567
 - higher-order iteration, 262
 - producer/consumer, 262
 - transducer, 263
- Ockham, William of, 31
- octal integer, 822
- Okasaki, Chris, 150, 178, 284, 335
- OLE (Object Linking and Embedding), 466
- OMG (Object Management Group), 466
- open distribution, 715, 717, 720
- open program, 205
- Open Source software, xxxviii
- operating system, 211
- operational semantics, 40, 61, 647, 783–814
- operator

- associativity, 36, 839
 - infix, 54, 84, 187, 796, 828, 830, 832, 838
 - mixfix, 827, 838
 - postfix, 839
 - precedence, 36, 838
 - prefix, 838
 - ternary, 841
- OPI (Oz Programming Interface), 817
- optimistic scheduling, 612
- optimization, 179
 - avoid during development, 455
 - combinatoric, 674
 - compiler, 164
 - eager execution, 308
 - early error detection, 510
 - first-argument indexing, 671
 - memoization, 703
 - monitor performance, 606
 - object system, 552
 - relational programming, 634
 - short-circuit protocol, 567
 - standard computer, 314
 - thread priorities, 270
- orelse** operator, 84
- otherwise** method, 506
- OTP (Ericsson Open Telecom Platform), 394
- overloading, 331
- Oz, Wizard of, 3
- ozc command, 232, 819
- Panel tool, *see* Mozart Programming System
- Papert, Seymour, xxvii, 237
- paradigm, xxvii, xxxi, *see* computation model
 - declarative, 32
 - school of thought, xxx
- parallelism, 241, 322
 - importance of non-strictness, 337
 - importance of worst-case complexity, 177
- parameter passing, 434–438
 - call by name, 437
 - exercise, 489
 - call by need, 437
 - exercise, 490
 - lazy evaluation, 438
 - call by reference, 59, 434
 - Java, 561
 - call by value, 435
 - Java, 561
 - call by value-result, 436
 - call by variable, 435
- parity, 16
- Parnas, David Lorge, xxxvi, 466
- parser, 34, 163–169
 - generic, 662
 - gump** tool, 41
 - natural language, 654
- partial termination, 247, 344, 806
- partial value, 47
- Pascal's triangle, 6
- Pascal, Blaise, 7
- pass by ..., *see* parameter passing
- pattern matching
 - case** statement, 8, 68
 - function (in Erlang), 396
 - Haskell, 327
 - receive** expression (in Erlang), 399
 - reduction rule semantics, 787
- pencil, xxxii
- Pentium III processor, 204, 475
- performance
 - cluster computing, 717
 - competitive concurrency, 259
 - constraint programming, 764
 - Cray-1 supercomputer, 178
 - declarative programming, 314
 - dictionary, 204
 - distributed stream, 730
 - how to measure, 170
 - lazy language, 295, 349
 - low-cost PC, 178
 - memoization, 27, 315
 - mobile object, 730
 - monitor, 606
 - Mozart Programming System, 204, 387
 - price of concurrency, 345
 - role of optimization, 179, 270, 308
 - role of parallelism, 241, 322
 - transitive closure, 474

- “Word of Mouth” simulation, 491
- permanent failure, 745
- permutations, 4
- persistence
 - data structure, 152, 303
 - database, 667
 - Erlang, 395
 - transaction, 609
- personal computer, 6, 178, 256, 259, 294, 309
- pessimistic scheduling, 612
- Phidani Software, 654
- π calculus, xxxi, 43, 55, 807
- pipeline, 263
- pixel, 562
- placeholder
 - dataflow variable, 88
 - future (in Multilisp), 342
 - GUI design, 696, 712
- planning, 633
- Plotkin, Gordon, 783
- point, 560
- POLA (Principle of Least Authority), 212
- polymorphism, 108, 466, 498
 - Haskell, 330
- Ponsard, Christophe, 551
- port (explicit state), 356–357, 725, 850
- portal, 481
- potential function, 178
- predicate calculus, 645
- preemption, 257
- preprocessor, 319
 - DCG (in Prolog), 662
 - design patterns, 542
 - extended DCG (in Prolog), 143
 - fallacy of, 319
- presentation model, 703
- principle
 - abstraction, 418
 - avoid changing interfaces, 462
 - avoid premature optimization, 179, 455
 - balance planning and refactoring, 455
 - centralized first, distributed later, 751
 - compartmentalize responsibility, 454
 - creative extension, xxviii, 846
 - decisions at right level, 464
 - declarative concurrency, 247, 286
 - document component interfaces, 454
 - documented violations, 464
 - eager default, lazy declared, 335
 - either simple or wrong, xl
 - encapsulate design decisions, 462
 - enriching control (in logic programming), 653
 - error confinement, 91
 - “everything should be an object”, 548
 - exploit ADT uniformity, 549
 - form mirrors content, 551
 - freely exchange knowledge, 454
 - layered language design, 852
 - least authority, 212
 - least privilege, 212
 - minimize dependencies, 463
 - minimize indirections, 463
 - model independence, 460
 - more is not better or worse, just different, xxxiv
 - need to know, 212
 - pay only on use, 631
 - predictable dependencies, 463
 - run time is all there is, 510
 - separation of concerns, 574
 - stateful component with declarative behavior, 425
 - syntax stability, 655
 - system decomposition, 213
 - use ADTs everywhere, 549
 - working software keeps working, 60, 463, 728
- private scope, 513, 514
 - C++ and Java sense, 514
 - Smalltalk and Oz sense, 513
- probability
 - Exponential distribution, 480
 - Gaussian distribution, 480
 - Normal distribution, 480
 - Uniform distribution, 478
 - unique name generation, 210
- proc** statement, 66, 795
- procedure