PROJECT SUGGESTIONS

The exercises in this book are intended to deepen your understanding of ML and improve your programming skills. But such exercises cannot turn you into a programmer, let alone a software engineer. A project is more than a large programming exercise; it involves more than programming. It demands careful preparation: background study, analysis of requirements, design. The finished program should be evaluated fairly but thoroughly.

Each suggestion is little better than a hint, but with a little effort, can be developed into a proper proposal. Follow the attached references and prepare a project description including a statement of objectives, a provisional timetable and a list of required resources. The next stage is to write a detailed requirements analysis, listing all functions in sufficient detail to allow someone else to carry out eventual testing. Then specify the basic design; ML functors and signatures can describe the main components and their interfaces.

The preparatory phases outlined above might be done by the instructor, a student or a team of students. This depends upon the course aims, which might be concerned purely with ML, with project management, or with demonstrating some methodology of software engineering. The final evaluation might similarly be done by the instructor, the implementor or another team of students.

The evaluation should consider to what extent the program meets its objectives. Testing can be driven by the requirements analysis. Many projects are easy to do, but hard to do efficiently. Evaluation thus should consider performance as well as correctness; profiling tools can identify performance bottlenecks. Students might be expected to find and use the standard library modules — arrays, low-level word operations, etc. — that are appropriate for efficiency.

Some of the suggested projects have been done by Cambridge students, though not necessarily in ML. The others are there because they are interesting (at least to me) and are of suitable difficulty. They are intended to be particularly appropriate for ML — though practically anything can be done in ML unless it requires unsafe programming or is embedded in a system that mandates some other language. So, feel free to adopt project suggestions from other sources.

Unlimited precision integer arithmetic gives exact answers and never fails due to overflow. (Some ML systems provide this by default.) Knuth (1981) describes the algorithms, which apart from division are straightforward. He also suggests improved algorithms for operating on rational numbers.

Unlimited precision real arithmetic yields answers that are correct to any desired precision, automatically determining the precision required for intermediate calculations. Much effort has been devoted to finding the most efficient representation of a real number (Boehm and Cartwright, 1990). Ménissier-Morain (1995) recommends a convergent series of rationals of the form p/B^q. Computational stunts on this theme may be amusing, though definitely not easy (Gourdon and Salvy, 1993). You could also develop the numerical examples of Section 5.15.

The ***polynomial arithmetic*** example of Chapter 3 can be extended in several directions. You could provide additional operations, allow more than one variable or even implement a better GCD algorithm. See Davenport *et al.* (1993) or Knuth (1981). Unlimited precision integers are required.

Emulators can be fun: you bring an obsolete machine and its quaint software back to life. My personal favourite is the DEC PDP-8. The basic model can address 4096 12-bit words and has an instruction set with eight opcodes. The manuals are out of print, but information is available on the World Wide Web (Jones, 1995). Details such as the precise treatment of interrupts can be tricky. Emulated software must run fast enough to keep up with the user's typing!

Advanced tautology checkers include ordered binary decision diagrams and the Davis-Putnam proof procedure. OBDDs have applications in hardware and systems verification; Bryant (1992) is a classic description but Moore (1994) may be more appropriate for functional programming. Davis-Putnam is back in favour after many years; early books mention it (Chang and Lee, 1973), but the latest algorithms are described only in technical reports (Zhang and Stickel, 1994).

Theorem provers can be built in various ways upon the foundation provided in Chapter 10. The tableau method is easy to implement (Beckert and Posegga, 1995). Model elimination is also fairly straightforward (Stickel, 1988a). Andrews (1989) describes the matrix method in the context of higher-order logic; it is equally applicable to first-order logic. Only the ablest student should try implementing the resolution method (Stickel, 1988b); the refinements necessary for high performance demand complex data structures (Butler and Overbeek, 1994).

Consider writing a ***parser generator***: a simple LR(0) one, an SLR one, or perhaps an LALR(1) version with sophisticated error recovery. Good compiler texts, such as Aho *et al.* (1986), describe the necessary techniques.

Compiling projects are always popular. Select a small subset of ML and write an interpreter for it. The SECD machine yields call-by-value semantics, while graph reduction yields call-by-need. Field and Harrison (1988) describe such implementation methods, as well as type checking. Unless the syntax is trivial, use a parser generator such as ML-Yacc (Tarditi and Appel, 1994).

You should be familiar at least with Chapters 2–5, preferably also Chapters 7 and 8, before attempting any substantial project. Good luck!

BIBLIOGRAPHY

Aasa, A., Holmström, S., and Nilsson, C. (1988). An efficiency comparison of some representations of purely functional arrays. *BIT*, **28**, 490–503.

Abelson, H. and Sussman, G. J. (1985). *Structure and Interpretation of Computer Programs*. MIT Press.

Adams, S. (1993). Efficient sets – a balancing act. *Journal of Functional Programming*, **3**(4), 553–561.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques and Tools*. Addison-Wesley.

Andrews, P. B. (1989). On connections and higher-order logic. *Journal of Automated Reasoning*, **5**(3), 257–291.

Appel, A. W. (1992). *Compiling with Continuations*. Cambridge University Press.

Appel, A. W. (1993). A critique of standard ML. *Journal of Functional Programming*, **3**(4), 391–429.

Appel, A. W., Mattson, J. S., and Tarditi, D. R. (1994). *A Lexical Analyzer Generator for Standard ML*, version 1.5.0 edition. Distributed with Standard ML of New Jersey and through freeware archives.

Augustsson, L. and Johnsson, T. (1989). The Chalmers Lazy-ML compiler. *Computer Journal*, **32**, 127–141.

Backus, J. (1978). Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, **21**, 613–641.

Bailey, R. (1990). *Functional Programming with Hope*. Ellis Horwood.

Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics*. North-Holland.

Beckert, B. and Posegga, J. (1995). leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, **15**(3), 339–358.

Bevier, W. R., Hunt, Jr., W. A., Moore, J. S., and Young, W. D. (1989). An approach to systems verification. *Journal of Automated Reasoning*, **5**(4), 411–428.

Biagioni, E., Harper, R., Lee, P., and Milnes, B. G. (1994). Signatures for a network protocol stack: A systems application of Standard ML. In *LISP and Functional Programming*, pages 55–64.

Bird, R. and Wadler, P. (1988). *Introduction to Functional Programming*. Prentice-Hall.

Boehm, H. and Cartwright, R. (1990). Exact real arithmetic: Formulating real numbers as functions. In Turner (1990b), pages 43–64.

Boolos, G. S. and Jeffrey, R. C. (1980). *Computability and Logic*. Cambridge University Press, 2nd edition.

Boyer, R. S. and Moore, J. S. (1988). *A Computational Logic Handbook*. Academic Press.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *Computing Surveys*, **24**(3), 293–318.

Burge, W. H. (1975). *Recursive Programming Techniques*. Addison-Wesley.

Burton, F. W. (1982). An efficient functional implementation of FIFO queues. *Information Processing Letters*, **14**, 205–206.

Butler, R. M. and Overbeek, R. A. (1994). Formula databases for high-performance resolution/paramodulation systems. *Journal of Automated Reasoning*, **12**(2), 139–156.

Cann, D. (1992). Retire Fortran? a debate rekindled. *Communications of the ACM*, **35**(8), 81–89.

Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, **17**, 471–522.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Cohn, A. (1989a). Correctness properties of the Viper block model: The second level. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 1–91. Springer.

Cohn, A. (1989b). The notion of proof in hardware verification. *Journal of Automated Reasoning*, **5**(2), 127–139.

Constable, R. L. *et al.* (1986). *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall.

Corbin, J. and Bidoit, M. (1983). A rehabilitation of Robinson's unification algorithm. In R. E. A. Mason, editor, *Information Processing 83*. IFIP, Elsevier.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

Cousineau, G. and Huet, G. (1990). The CAML primer. Technical report, INRIA, Rocquencourt, France.

Damas, L. and Milner, R. (1982). Principal type-schemes for functional programs. In *9th Symposium on Principles of Programming Languages*, pages 207–212. ACM.

Davenport, H. (1952). *The Higher Arithmetic*. Cambridge University Press, 6th edition.

Davenport, J. H., Siret, Y., and Tournier, E. (1993). *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press.

de Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indagationes Mathematicae*, **34**, 381–392.

Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice-Hall.

Feynman, R. P., Leighton, R. B., and Sands, M. (1963). *The Feynman Lectures on Physics*, volume 1. Addison-Wesley.

Field, A. J. and Harrison, P. G. (1988). *Functional Programming*. Addison-Wesley.

Fitzgerald, J. S., Larsen, P. G., Brookes, T. M., and Green, M. A. (1995). Developing a security-critical system using formal and conventional methods. In Hinchey and Bowen (1995), pages 333–356.

FPCA (1995). *Functional Programming and Computer Architecture*. ACM Press.

Frost, R. and Launchbury, J. (1989). Constructing natural language interpreters in a lazy functional language. *Computer Journal*, **32**, 108–121.

Gallier, J. H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row.

Galton, A. (1990). *Logic for Information Technology*. Wiley.

Gansner, E. R. and Reppy, J. H., editors (1996). *The Standard ML Basis Library Reference Manual*. In preparation.

Gerhart, S., Craigen, D., and Ralston, T. (1994). Experience with formal methods in critical systems. *IEEE Software*, pages 21–28.

Gordon, M. J. C. (1988). *Programming Language Theory and its Implementation*. Prentice-Hall.

Gordon, M. J. C. and Melham, T. F. (1993). *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press.

Gordon, M. J. C., Milner, R., and Wadsworth, C. P. (1979). *Edinburgh LCF: A Mechanised Logic of Computation*. LNCS 78. Springer.

Gourdon, X. and Salvy, B. (1993). Computing one million digits of $\sqrt{2}$. Technical Report 155, INRIA-Rocquencourt.

Graham, B. T. (1992). *The SECD Microprocessor: A Verification Case Study*. Kluwer Academic Publishers.

Grant, P. W., Sharp, J. A., Webster, M. F., and Zhang, X. (1995). Experiences of parallelising finite-element problems in a functional style. *Software—Practice and Experience*, **25**(9), 947–974.

Grant, P. W., Sharp, J. A., Webster, M. F., and Zhang, X. (1996). Sparse matrix representations in a functional language. *Journal of Functional Programming*, **6**(1), 143–170.

Greiner, J. (1996). Weak polymorphism can be sound. *Journal of Functional Programming*, **6**(1), 111–141.

Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. MIT Press.

Halfant, M. and Sussman, G. J. (1988). Abstraction in numerical methods. In *LISP and Functional Programming*, pages 1–7. ACM Press.

Harper, R. (1994). A simplified account of polymorphic references. *Information Processing Letters*, **51**(4), 201–206.

Harper, R., MacQueen, D., and Milner, R. (1986). Standard ML. Technical Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh.

Hartel, P. and Plasmeijer, R. (1996). State-of-the-art applications of pure functional programming. *Journal of Functional Programming*, **5**(3). Special issue.

Hennessy, M. (1990). *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics*. Wiley.

Hinchey, M. and Bowen, J. P., editors (1995). *Applications of Formal Methods*. Prentice-Hall.

Hoare, C. A. R. (1989a). Computer science. In Hoare and Jones (1989), pages 89–101. Inaugural lecture, Queen's University of Belfast, 1971.

Hoare, C. A. R. (1989b). Hints on programming-language design. In Hoare and Jones (1989), pages 193–216. First appeared in 1974.

Hoare, C. A. R. (1989c). An overview of some formal methods for program design. In Hoare and Jones (1989), pages 371–387. Reprinted from *Computer* **20** (1987), 85–91.

Hoare, C. A. R. and Jones, C. B., editors (1989). *Essays in Computing Science*. Prentice-Hall.

Hoogerwoord, R. (1992). A logarithmic implementation of flexible arrays. In R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, editors, *Mathematics of Program Construction: Second International Conference*, LNCS 669, pages 191–207. Springer.

Hudak, P., Jones, S. P., and Wadler, P. (1992). Report on the programming language Haskell: A non-strict, purely functional language. *SIGPLAN Notices*, **27**(5). Version 1.2.

Huet, G. P. (1975). A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, **1**, 27–57.

Hughes, J. (1989). Why functional programming matters. *Computer Journal*, **32**, 98–107.

Jones, D. W. (1995). The Digital Equipment Corporation PDP-8. On the World Wide Web at URL `http://www.cs.uiowa.edu/~jones/pdp8/`.

Kennedy, A. (1996). Functional pearls: Drawing trees. *Journal of Functional Programming*, **6**(3), 527–534.

Knuth, D. E. (1973). *The Art of Computer Programming*, volume 3: *Sorting and Searching*. Addison-Wesley.

Knuth, D. E. (1981). *The Art of Computer Programming*, volume 2: *Seminumerical Algorithms*. Addison-Wesley, 2nd edition.

Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, **27**, 97–109.

Lakatos, I. (1976). *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

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

Leroy, X. and Mauny, M. (1993). Dynamics in ML. *Journal of Functional Programming*, **3**(4), 431–463.

Letz, R., Schumann, J., Bayerl, S., and Bibel, W. (1992). SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, **8**(2), 183–212.

Magnusson, L. and Nordström, B. (published 1994). The ALF proof editor and its proof engine. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs: International Workshop TYPES '93*, LNCS 806, pages 213–237. Springer.

Martelli, A. and Montanari, U. (1982). An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, **4**, 258–282.

Mattson, Jr., H. F. (1993). *Discrete Mathematics with Applications*. Wiley.

McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., and Levin, M. I. (1962). *LISP 1.5 Programmer's Manual*. MIT Press.

Ménissier-Morain, V. (1995). Arbitrary precision real arithmetic: design and algorithms. Submitted to *Journal of Symbolic Computation*.

Mills, H. D. and Linger, R. C. (1986). Data structured programming: Program design without arrays and pointers. *IEEE Transactions on Software Engineering*, **SE-12**, 192–197.

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

Milner, R. and Tofte, M. (1990). *Commentary on Standard ML*. MIT Press.

Milner, R., Tofte, M., and Harper, R. (1990). *The Definition of Standard ML*. MIT Press.

Moore, J. S. (1994). Introduction to the OBDD algorithm for the ATP community. *Journal of Automated Reasoning*, **12**(1), 33–46.

Nederpelt, R. P., Geuvers, J. H., and de Vrijer, R. C., editors (1994). *Selected Papers on Automath*. North-Holland.

Odersky, M., Wadler, P., and Wehr, M. (1995). A second look at overloading. In FPCA (1995), pages 135–146.

Ohori, A. (1995). A polymorphic record calculus and its compilation. *ACM Transactions on Programming Languages and Systems*, **17**(6), 844–895.

Okasaki, C. (1995). Purely functional random-access lists. In FPCA (1995), pages 86–95.

O'Keefe, R. A. (1982). A smooth applicative merge sort. Research paper 182, Department of AI, University of Edinburgh.

Oppacher, F. and Suen, E. (1988). HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, **4**(1), 69–100.

Oppen, D. C. (1980). Pretty printing. *ACM Transactions on Programming Languages and Systems*, **2**, 465–483.

Park, S. K. and Miller, K. W. (1988). Random number generators: Good ones are hard to find. *Communications of the ACM*, **31**, 1192–1201.

Paterson, M. S. and Wegman, M. N. (1978). Linear unification. *Journal of Computer and System Sciences*, **16**, 158–167.

Paulson, L. C. (1983). A higher-order implementation of rewriting. *Science of Computer Programming*, **3**, 119–149.

Paulson, L. C. (1987). *Logic and Computation: Interactive proof with Cambridge LCF*. Cambridge University Press.

Paulson, L. C. (1994). *Isabelle: A Generic Theorem Prover*. Springer. LNCS 828.

Paulson, L. C. (1995). Set theory for verification: II. Induction and recursion. *Journal of Automated Reasoning*, **15**(2), 167–215.

Pelletier, F. J. (1986). Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, **2**, 191–216. Errata, JAR 4 (1988), 235–236.

Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press.

Peyton Jones, S. L. (1992). Implementing lazy functional languages on stock hardware: The spineless tagless G-machine. *Journal of Functional Programming*, **2**(2), 127–202.

Peyton Jones, S. L. and Wadler, P. (1993). Imperative functional programming. In *20th Symposium on Principles of Programming Languages*, pages 71–84. ACM Press.

Quaife, A. (1992). Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, **8**(1), 91–147.

Reade, C. (1989). *Elements of Functional Programming*. Addison-Wesley.

Reade, C. (1992). Balanced trees with removals: an exercise in rewriting and proof. *Science of Computer Programming*, **18**, 181–204.

Reeves, S. and Clarke, M. (1990). *Logic for Computer Science*. Addison-Wesley.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, 2nd edition.

Ružička, P. and Prívara, I. (1988). An almost linear Robinson unification algorithm. In M. P. Chytil, L. Janiga, and V. Koubeck, editors, *Mathematical Foundations of Computer Science*, LNCS 324, pages 501–511. Springer.

Sedgewick, R. (1988). *Algorithms*. Addison-Wesley, 2nd edition.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM*, **32**(3), 652–686.

Smith, M. H., Garigliano, R., Morgan, R., Shiu, S., and Jarvis, S. (1994). LOLITA : A natural language engineered system. Technical report, Department of Computer Science, University of Durham.

Spafford, E. H. (1989). The Internet worm: Crisis and aftermath. *Communications of the ACM*, **32**(6), 678–687.

Srivas, M. K. and Miller, S. P. (1995). Formal verification of the AAMP5 microprocessor. In Hinchey and Bowen (1995), pages 125–180.

Stickel, M. E. (1988a). A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, **4**(4), 353–380.

Stickel, M. E. (1988b). Resolution theorem proving. *Annual Review of Computer Science*, **3**, 285–316.

Tarditi, D. R. and Appel, A. W. (1994). *ML-Yacc User's Manual*, version 2.2 edition. Distributed with Standard ML of New Jersey and through freeware archives.

Thompson, S. (1991). *Type Theory and Functional Programming*. Addison-Wesley.

Tofte, M. (1990). Type inference for polymorphic references. *Information and Computation*, **89**, 1–34.

Turner, D. (1990a). An overview of Miranda. In Turner (1990b), pages 1–16.

Turner, D., editor (1990b). *Research Topics in Functional Programming*. Addison-Wesley.

Turner, D. A. (1979). A new implementation technique for applicative languages. *Software—Practice and Experience*, **9**, 31–49.

Turner, R. (1991). *Constructive Foundations for Functional Languages*. McGraw-Hill.

Uribe, T. E. and Stickel, M. E. (1994). Ordered binary decision diagrams and the Davis-Putnam procedure. In J. P. Jouannaud, editor, *Constraints in Computational Logics: First International Conference*, LNCS 845, pages 34–49. Springer.

Wadler, P. and Gill, A. (1995). Real world applications of functional programming. On the World Wide Web at URL `http://www.dcs.gla.ac.uk/fp/realworld/`.

Wiener, L. R. (1993). *Digital Woes: Why We Should Not Depend on Software*. Addison-Wesley.

Winskel, G. (1993). *The Formal Semantics of Programming Languages*. MIT Press.

Wirth, N. (1985). *Programming in Modula-2*. Springer, 3rd edition.

Wright, A. K. (1995). Simple imperative polymorphism. *Lisp and Symbolic Computation*, **8**(4), 343–356.

Zhang, H. and Stickel, M. E. (1994). An efficient algorithm for unit propagation. Technical Report 94-12, Computer Science Dept., University of Iowa.