Introduction to Functional Programming

John Harrison
jrh@cl.cam.ac.uk

3rd December 1997

Preface

These are the lecture notes accompanying the course *Introduction to Functional Programming*, which I taught at Cambridge University in the academic year 1996/7.

This course has mainly been taught in previous years by Mike Gordon. I have retained the basic structure of his course, with a blend of theory and practice, and have borrowed heavily in what follows from his own lecture notes, available in book form as Part II of (Gordon 1988). I have also been influenced by those who have taught related courses here, such as Andy Gordon and Larry Paulson and, in the chapter on types, by Andy Pitts's course on the subject.

The large chapter on examples is not directly examinable, though studying it should improve the reader's grasp of the early parts and give a better idea about how ML is actually used.

Most chapters include some exercises, either invented specially for this course or taken from various sources. They are normally intended to require a little thought, rather than just being routine drill. Those I consider fairly difficult are marked with a (*).

These notes have not yet been tested extensively and no doubt contain various errors and obscurities. I would be grateful for constructive criticism from any readers.

John Harrison (`jrh@cl.cam.ac.uk`).

Plan of the lectures

This chapter indicates roughly how the material is to be distributed over a course of twelve lectures, each of slightly less than one hour.

1. **Introduction and Overview** Functional and imperative programming: contrast, pros and cons. General structure of the course: how lambda calculus turns out to be a general programming language. Lambda notation: how it clarifies variable binding and provides a general analysis of mathematical notation. Currying. Russell's paradox.

2. **Lambda calculus as a formal system** Free and bound variables. Substitution. Conversion rules. Lambda equality. Extensionality. Reduction and reduction strategies. The Church-Rosser theorem: statement and consequences. Combinators.

3. **Lambda calculus as a programming language** Computability background; Turing completeness (no proof). Representing data and basic operations: truth values, pairs and tuples, natural numbers. The predecessor operation. Writing recursive functions: fixed point combinators. Let expressions. Lambda calculus as a declarative language.

4. **Types** Why types? Answers from programming and logic. Simply typed lambda calculus. Church and Curry typing. Let polymorphism. Most general types and Milner's algorithm. Strong normalization (no proof), and its negative consequences for Turing completeness. Adding a recursion operator.

5. **ML** ML as typed lambda calculus with eager evaluation. Details of evaluation strategy. The conditional. The ML family. Practicalities of interacting with ML. Writing functions. Bindings and declarations. Recursive and polymorphic functions. Comparison of functions.

6. **Details of ML** More about interaction with ML. Loading from files. Comments. Basic data types: unit, booleans, numbers and strings. Built-in operations. Concrete syntax and infixes. More examples. Recursive types and pattern matching. Examples: lists and recursive functions on lists.

7. **Proving programs correct** The correctness problem. Testing and verification. The limits of verification. Functional programs as mathematical objects. Examples of program proofs: exponential, GCD, append and reverse.

8. **Effective ML** Using standard combinators. List iteration and other useful combinators; examples. Tail recursion and accumulators; why tail recursion is more efficient. Forcing evaluation. Minimizing consing. More efficient reversal. Use of 'as'. Imperative features: exceptions, references, arrays and sequencing. Imperative features and types; the value restriction.

9. **ML examples I: symbolic differentiation** Symbolic computation. Data representation. Operator precedence. Association lists. Prettyprinting expressions. Installing the printer. Differentiation. Simplification. The problem of the 'right' simplification.

10. **ML examples II: recursive descent parsing** Grammars and the parsing problem. Fixing ambiguity. Recursive descent. Parsers in ML. Parser combinators; examples. Lexical analysis using the same techniques. A parser for terms. Automating precedence parsing. Avoiding backtracking. Comparison with other techniques.

11. **ML examples III: exact real arithmetic** Real numbers and finite representations. Real numbers as programs or functions. Our representation of reals. Arbitrary precision integers. Injecting integers into the reals. Negation and absolute value. Addition; the importance of rounding division. Multiplication and division by integers. General multiplication. Inverse and division. Ordering and equality. Testing. Avoiding reevaluation through memo functions.

12. **ML examples IV: Prolog and theorem proving** Prolog terms. Case-sensitive lexing. Parsing and printing, including list syntax. Unification. Backtracking search. Prolog examples. Prolog-style theorem proving. Manipulating formulas; negation normal form. Basic prover; the use of continuations. Examples: Pelletier problems and whodunit.

Contents