Fast and fun results: functional programming for mathematicians

Mark Wildon

- Please ask questions.
- Full solutions to all problems are on my website: see www.ma.rhul.ac.uk/~uvah099/Talks/FuncProg.nb.
- Any comments or suggestions for things you’d have liked to see covered are very welcome.
- Do not try to use the free online version of Mathematica for the exercises. It is very slow and buggy.
“What I mean is that if you really want to understand something, the best way is to try and explain it to someone else. That forces you to sort it out in your own mind. And the more slow and dim-witted your pupil, the more you have to break things down into more and more simple ideas. And that’s really the essence of programming. By the time you’ve sorted out a complicated idea into little steps that even a stupid machine can deal with, you’ve certainly learned something about it yourself.”

Douglas Adams, Dirk Gently’s Holistic Detective Agency (1987)
“What I mean is that if you really want to understand something, the best way is to try and explain it to someone else. That forces you to sort it out in your own mind. And the more slow and dim-witted your pupil, the more you have to break things down into more and more simple ideas. And that’s really the essence of programming. By the time you’ve sorted out a complicated idea into little steps that even a stupid machine can deal with, you’ve certainly learned something about it yourself.”


I . . . am rarely happier than when spending an entire day programming my computer to perform automatically a task that would otherwise take me a good ten seconds to do by hand.

“What I mean is that if you really want to understand something, the best way is to try and explain it to someone else. That forces you to sort it out in your own mind. And the more slow and dim-witted your pupil, the more you have to break things down into more and more simple ideas. And that’s really the essence of programming. By the time you’ve sorted out a complicated idea into little steps that even a stupid machine can deal with, you’ve certainly learned something about it yourself.”


I . . . am rarely happier than when spending an entire day programming my computer to perform automatically a task that would otherwise take me a good ten seconds to do by hand.


“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil”

Some programming paradigms

- Imperative (for example, C)
  
  ```c
  int f(int n) {
    int a = 0; int b = 1; int c;
    int i; for (i = 0; i < n; i++) {
      c = a + b; a = b; b = c;
    }
    return a;
  }
  ```

- Functional (for example, Haskell)
  
  ```haskell
  f 0 = 0; f 1 = 1; f n = f (n-1) + f (n-2);
  ```

- Rule-based (for example, Inform 7)
  
  The description of the notepad is “A normal notepad. On it you see written [15 th Fibonacci number].”
  
  Definition: a number is small if it less than 2.
  
  To decide which number is the (n - a number) th Fibonacci number: if n is a small number then decide on n; otherwise decide on the (n - 1) th Fibonacci number plus the (n - 2) th Fibonacci number.
Mathematica supports all three paradigms

- It is fastest and most elegant when used as a functional programming language.
- Pattern matching and transformation rules allow use of rule-based programming.

Promise: you will be able to solve all problems today using only Mathematica functions introduced in this talk.

- Functions: square brackets. For instance
  
  fib[0] := 0
  fib[1] := 1
  fib[n_] := fib[n-1] + fib[n-2]

  := is syntactic sugar for SetDelayed. The right-hand side is stored in Mathematica’s internal memory, and evaluated when necessary.

  n_ is a pattern, matching anything. Whatever it matches, will be used in place of ‘n’ on the right-hand side.

- Slow? See final slide on memoization: won’t be covered today.

- To find out what is stored under a symbol, for instance f, use DownValues[f]. To clear the definition use Clear[f].
A toy RSA-cryptosystem

Dr Z, a somewhat naïve pure mathematician, has chosen as his RSA modulus

\[ \text{NextPrime}[2^{32}+2^{31}] \times \text{NextPrime}[2^{32}+2^{16}] \]

and decides on \( e = 11 \) as a convenient choice of encryption exponent.

- Write Mathematica functions Encrypt and Decrypt for communication of a number between 0 and \( 2^{64} \).
  - Useful functions: \( \text{Mod}[n,p] \) returns \( n \mod m \), \( \text{PowerMod}[n,-1,m] \) returns \( n^{-1} \mod m \).
  - Mathematica has all the usual calculator functions, +, −, ×, exponentiation . . .
  - \( \text{If}[\text{cond},x,y] \) is \( x \) if \( \text{cond} \) is true, \( y \) if \( \text{cond} \) is false.

Discussion: give some of the ways in which Dr Z's cryptosystem might be improved.

- Write an efficient function using only \( \text{Mod}, \text{If} \) and recursion that computes \( x^e \mod n \) for any \( x, e, n \in \mathbb{N} \). Helpful function: \( \text{EvenQ}[e] \) is True if \( e \) is even, otherwise False.
A toy RSA-cryptosystem

Dr Z, a somewhat naïve pure mathematician, has chosen as his RSA modulus

\[ \text{NextPrime}[2^{32}+2^{31}] \times \text{NextPrime}[2^{32}+2^{16}] \]

and decides on \( e = 11 \) as a convenient choice of encryption exponent.

- Write Mathematica functions Encrypt and Decrypt for communication of a number between 0 and \( 2^{64} \).

  - Useful functions: \( \text{Mod}[n,p] \) returns \( n \mod m \), \( \text{PowerMod}[n,-1,m] \) returns \( n^{-1} \mod m \).
  - Mathematica has all the usual calculator functions, +, −, ×, exponentiation . . .
  - \( \text{If}[\text{cond},x,y] \) is \( x \) if \( \text{cond} \) is true, \( y \) if \( \text{cond} \) is false.

Discussion: give some of the ways in which Dr Z's cryptosystem might be improved.

- Write an efficient function using only \( \text{Mod}, \; \text{If} \) and recursion that computes \( x^e \mod n \) for any \( x, \; e, \; n \in \mathbb{N} \). Helpful function: \( \text{EvenQ}[e] \) is True if \( e \) is even, otherwise False.
Lists and map/reduce

- Map[f, xs] evaluates f on each member of the list xs. For example, Map[fib, {1, 2, 3}] ↦ {f[1], f[2], f[3]} ↦ {1, 1, 2}. The symbol # is an anonymous argument: for example Map[#*2 &, xs] doubles every element of xs.

- Select[xs, pred] selects those elements of the list xs satisfying the predicate pred. For example, Select[{1, 2, 3}, OddQ] ↦ {1, 3}.

- The ‘FullForm’ representation of {1, 2, 3} is List[1, 2, 3]. Apply replaces the head ‘List’ with another function of your choice. For example Apply[Plus, {1, 2, 3}] ↦ 6.

Some other useful functions.

- x == y ↦ True if x and y are the same, ↦ False otherwise.
- Range[1, 4] ↦ {1, 2, 3, 4}.
- Join[{{1, 2, 3}, {1, 2}, {1}}, {1}] ↦ {1, 2, 3, 1, 2, 1}.
- Table[f[x], {x, ys}] ↦ Map[f, ys]

- Write functions EncryptList and DecryptList that allow Dr Z’s cryptoscheme to communicate longer messages.
Further map/reduce problems

- Write a function that returns True if and only if its input is a list of odd numbers using And. For example
  \(\text{And}[\text{True}, \text{False}, \text{True}] \Rightarrow \text{False}\).

- Write a function \text{CountList} that given a list of numbers, returns a list of pairs counting the number of appearances of each number. For example
  \[
  \text{CountList}[\{1, 5, 2, 1, 2, 1\}]
  \]
  should evaluate to
  \[
  \{\{1, 3\}, \{5, 1\}, \{2, 2\}\}
  \]
  Useful functions: \text{First}[\text{xs}] returns the first element of the non-empty list \text{xs}, \text{Drop}[\text{xs}, 1] removes the first element, \text{Length}[\text{xs}] evaluates to the length of \text{xs}.

- Investigate asymptotics of \(\sum_{k=1}^{n} \frac{\phi(k)}{k}\). Useful functions: \text{EulerPhi}, \text{N} (numerical eval.), \text{TableForm} (format table).

- To mergesort a list, split it into two halves, mergesort each half, and then merge the lists back together. For example, the sorted lists \{4, 4, 6\} and \{1, 4, 5\} merge to \{1, 4, 4, 4, 5, 6\}. Write a \text{Mergesort} function. Useful function: \text{Take}. 
Quiz: with these definitions,

\[
\begin{align*}
  f[1] & := 1 \\
  f[x] & := x+1 \\
  f[y_\_] & := y/2
\end{align*}
\]
how will \( f[1] \), \( f[x+1] \), \( f[x] \), \( f[y] \), \( f[z] \), \( f[{1,2}] \) evaluate?
Patterns

Quiz: with these definitions,

\[
\begin{align*}
  f[1] & := 1 \\
  f[x] & := x+1 \\
  f[y_] & := y/2
\end{align*}
\]

how will \( f[1] \), \( f[x+1] \), \( f[x] \), \( f[y] \), \( f[z] \), \( f[{1,2}] \) evaluate?

The rule ‘most specific pattern wins’ was used to define \( \text{fib} \).

▶ Write a function to compare two lists under the lexicographic order.

Cases, ReplaceAll (or /. ) and Condition (or /;).

▶ \( \text{Cases}[\{\{1,2\}, 2, 3, \{3\}, \{4,\{5,6\}\}\}, \{\_, \_\}] \leadsto \{\{1,2\}, \{4,\{5,6\}\}\} \)

▶ \( \{1,2,3,\{4,5\}\} /. \{x_ \rightarrow x+1\} \leadsto \{2,3,4,\{5,6\}\} \)

▶ \( \{1,2,3,\{4,5\}\} /. \{x_ \;/; (x < 3) \rightarrow x+1\} \leadsto \{2,3,3,\{4,5\}\} \)

The next exercise is hard for annoying reasons.

▶ Write a function \( \text{PlotDerivative} \) to plot the derivative of a given function \( g \) of one variable.
Derangements

A derangement of the set \( \{1, 2, \ldots, n\} \) is a permutation \( f : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\} \) such that \( f(k) \neq k \) for any \( k \). In Mathematica, we will represent \( f \) by the list with elements \( f(1), f(2), \ldots, f(n) \).

- Write a function `IsDerangement` to decide if a permutation of \( \{1, 2, \ldots, n\} \), represented by a Mathematica list, is a derangement.
- Write a function `NumberOfDerangements` giving the number \( d_n \) of derangements of \( \{1, 2, \ldots, n\} \). [Hint: use `Permutations` to get all permutations.]
- Investigate the asymptotics of \( d_n \).
- Write a function to compose two permutations.
- Investigate the asymptotic probability that the composition of two derangements is a derangement.
Set and memoization

So far we have always used :=, or in ‘FullForm’, SetDelayed, for assignment. Sometimes it is useful to evaluate the right-hand immediately.

This is done used =, or in ‘FullForm’, Set.

When \( x = y \) is evaluated, \( y \) is evaluated, and the result assigned to \( x \); the return value is the evaluation of \( y \).

▶ Memoization: the Fibonacci function defined earlier uses exponentially many evaluations. For instance \( \text{Fib}[5] \mapsto \text{Fib}[4] + \text{Fib}[3] \), and then \( \text{Fib}[4] \mapsto \text{Fib}[3] + \text{Fib}[2] \) and \( \text{Fib}[3] \mapsto \text{Fib}[2] + \text{Fib}[1] \), so already we see \( \text{Fib}[2] \) will be evaluated twice.

▶ What we need is to force the evaluation of \( \text{Fib}[4] \) and \( \text{Fib}[3] \) and then store the result once and for all in \( \text{Fib}[5] \). Set is ideal for this.

▶ \( \text{fib}[0] := 0 \)
\( \text{fib}[1] := 1 \)
\( \text{fib}[n_] := \text{fib}[n] = \text{fib}[n-1] + \text{fib}[n-2] \)
Examples from research

- Foulkes’ Conjecture: Haskell implementation of new recurrence. Visualizing data: Haskell program `plotter.hs` produces Metafont files, which are turned into postscript files by Metafont, and finally printed or viewed as pdf.

- Derangements: new numerical results obtained using MAGMA and Haskell.