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.
- **Remember:** Shift-Return after each input line. It you just press return MATHEMATICA will not evaluate it.

“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."

Douglas Adams, *Dirk Gently's Holistic Detective Agency* (1987)

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.

Douglas Adams, *Last chance to see* (1989)

"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)

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.

Douglas Adams, *Last chance to see* (1989)

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

Donald Knuth, *Structured Programming with go to Statements* (1974)

Some programming paradigms

- Imperative (for example, 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)

```
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 is less than 2.

  To decide which number is the (n - a number) th Fibonacci number: if n is small, 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 can be very powerful.

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. Most specific pattern wins: so first line is used for `fib[0]`. Ties are broken by the order of input: sometimes it is essential to get this right (see `PowerMod` example below).
  - **Slow?** See final slide on memoization.

Patterns

- To find out what is stored under a symbol, for instance `fib`, use `Information[fib]`. Clear using `ClearAll[fib]`.

If you only want a pattern to match if an extra condition holds, use a pattern guard. For example

```
Collatz[x_] /; EvenQ[x] := x/2
Collatz[x_] /; OddQ[x]  := 3x+1
```

defines the Collatz function.

Quiz: with these definitions,

```
g[1]  := 1
g[x]  := x+1
g[y_] := y/2
g[{x_,y_}] /; EvenQ[y] := y/2
```

how will `g[1]`, `g[x+1]`, `g[x]`, `g[y]`, `g[z]`, `g[{1,2}]` evaluate?

Patterns

- To find out what is stored under a symbol, for instance `fib`, use `Information[fib]`. Clear using `ClearAll[fib]`.

If you only want a pattern to match if an extra condition holds, use a pattern guard. For example

```
Collatz[x_] /; EvenQ[x] := x/2
Collatz[x_] /; OddQ[x]  := 3x+1
```

defines the Collatz function.

Quiz: with these definitions,

```
g[1]   := 1
g[x]   := x+1
g[y_] := y/2
g[{x_,y_}] /; EvenQ[y] := y/2
```

how will `g[1]`, `g[x+1]`, `g[x]`, `g[y]`, `g[z]`, `g[{1,2}]` evaluate?

Remember: the most specific pattern wins.

Examples from teaching

- This term I've lectured MT362 Cipher Systems. MATHEMATICA was useful for implementing the old-school attacks on alphabetical ciphers. Using Haskell I implemented differential and linear cryptanalysis attacks on block ciphers.

Examples from teaching

- For a project student classifying all groups with exactly 4 conjugacy classes, we needed the solutions to

$$\frac{1}{n_1} + \frac{1}{n_2} + \frac{1}{n_3} + \frac{1}{n_4} = 1$$

with $n_1 \geq n_2 \geq n_3 \geq n_4$. By elementary inequalitites, either $n_1 = n_2 = n_3 = n_4 = 4$ or $n_4 \leq 3$, $n_3 \leq 6$ and $n_2 \leq 12$, giving only finitely many solutions to search for.

```
 GoodTriple[{y2_, y3_, y4_}] :=
      And[y2 + y3 + y4 < 1, y2 <= y3, y3 <= y4,
      1 - y2 - y3 - y4 <= y2,
      IntegerQ[1/(1 - y2 - y3 - y4)]]
Select[Join @@ Join @@ Table[{1/n2, 1/n3, 1/n4},
      {n4, {2, 3}}, {n3, {2, 3, 4, 5, 6}},
      {n2, Range[2, 12]}], GoodTriple]
```

In the functional style we define `GoodTriple` to recognise solutions and use `Select` to apply it to all the candidates, built using `Table`.

A toy RSA-Cryptosystem

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

```
NextPrime[2^32+2^31]*NextPrime[2^32+2^16]
```

and decides on $e = 2$ as his encryption exponent.

- Write MATHEMATICA functions `ToyEncrypt` and `ToyDecrypt` to encrypt and decrypt arbitrary numbers in this scheme. (Expect to find a problem with `ToyDecrypt`.)
  - Useful functions: `Mod[x,p]` returns x mod m, `PowerMod[x,-1,m]` returns x^{-1} mod m.
  - MATHEMATICA has all the usual calculator functions, $+$, $-$, $\times$, exponentiation ...
  - `If[cond,x,y]` is `x` if `cond` is `true`, `y` if `cond` is false.

A toy RSA-Cryptosystem

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

```
NextPrime[2^32+2^31]*NextPrime[2^32+2^16]
```

and decides on $e = 2$ as his encryption exponent.

- Write MATHEMATICA functions `ToyEncrypt` and `ToyDecrypt` to encrypt and decrypt arbitrary numbers in this scheme. (Expect to find a problem with `ToyDecrypt`.)
  - Useful functions: `Mod[x,p]` returns x mod m, `PowerMod[x,-1,m]` returns x^{-1} mod m.
  - MATHEMATICA has all the usual calculator functions, $+$, $-$, $\times$, exponentiation ...
  - `If[cond,x,y]` is `x` if `cond` is `true`, `y` if `cond` is false.

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

- Write an efficient function using only `Mod`, `If` (or pattern guards) and recursion that computes x^e mod n for any $x, e, n \in \mathbf{N}$.

PowerMod Example

A solution using pattern guards is

```
PM[x_, 0, n_] := 1
PM[x_, e_, n_] /; EvenQ[e] := PM[Mod[x^2, n], e/2, n]
PM[x_, e_, n_] /; OddQ[e]  :=
              Mod[x*PM[Mod[x, n], e-1, n], n]
```

Because of the pattern guards in the second and third cases, MATHEMATICA not consider the first rule as the most specific. So it is essential to enter it first to give the recursion a base case.

A solution using `If` is

```
PMIf[x_,0,n_] := 1
PMIf[x_, e_, n_] :=
     If[EvenQ[e], PM[z, e/2, n],
                  Mod[x*PMIf[z, e-1, n], n]]
```

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}]` ⇝ `{1,2,3,1,2,1}`.
- `Table[f[x],{x,ys}]` ⇝ `Map[f,ys]`

Map exercise

Dr Z decides it would be nice to be able to send English messages rather than just numbers.

Quiz: Given that

```
ToCharacterCode["H"] ⇝ 72
ToCharacterCode["E"] ⇝ 69
ToCharacterCode["L"] ⇝ 76
```

what is

```
Map[ToCharacterCode, {"H","E","L","L","O"}] ⇝ ?
```

Write functions `ToyEncryptWord` and `ToyDecryptWord`. Hint: glue together simpler functions. So `ToyEncryptWord` could be the composition of `WordToNumbers` and `ToyEncryptNumbers`.

- Write a function that computes the sum $s(n)$ of the (base 10) digits of a number n. Useful functions:
  - `Mod[x,10]`, `Quotient[x,10]`
- Define $S : \mathbf{N} \to \mathbf{N}$ by

$$S(n) = \begin{cases} n & \text{if } n < 10 \\ S(s(n)) & \text{if } n \geq 10. \end{cases}$$

  Why is S well-defined? Implement S in MATHEMATICA.
  - There are solutions using `If` or pattern guards.
  - A very elegant solution uses `//.` (apply rule repeatedly until there is no change).
- Investigate $S(n^2)$ for $n \in \mathbf{N}$: `Table[S[x^2],{x,1,10}]`
- Multiple iterators give nested lists. This is not always what one wants. For example `Table[i+j,{i,1,2},{j,1,2}]` $\rightsquigarrow$ `{{2, 3}, {3, 4}}`. Instead use

  `Join@@Table[i+j,{i,1,2},{j,1,2}]` $\rightsquigarrow$ `{2,3,3,4}`.
  - Challenge: make all lists of a given length from a given list: `Ls[{1,2},2]` $\rightsquigarrow$ `{{1, 1}, {1, 2}, {2, 1}, {2, 2}}`.

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 And[True,False,True] $\rightsquigarrow$ False.
- Write a function CountList that given a list of numbers, returns a list of pairs counting the number of appearances of each number. For example

  CountList[{1,5,2,1,2,1}]

  should evaluate to

  {{1,3},{5,1},{2,2}}

  Useful functions: First[xs] returns the first element of the non-empty list xs, Drop[xs,1] removes the first element, Length[xs] evaluates to the length of xs.
- Investigate asymptotics of $\sum_{k=1}^{n} \phi(k)/k$. Useful functions: EulerPhi, N (numerical eval.), 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 Mergesort function. Useful function: Take.

Pattern Exercises

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

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

- `Cases[{{1, 2}, 2, 3, {3},{4,{5,6}}}, {_, _}]` ⇝ `{{1,2},{4,{5,6}}}`
- `{1,2,3,{4,5}} /. {x_ :> x+1}` ⇝ `{2,3,4,{5,6}}`
- `{1,2,3,{4,5}} /. {x_ /; (x < 3) :> x+1}` ⇝ `{2,3,3,{4,5}}`

The next exercise is hard for annoying reasons.

- Write a function `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 $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ 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.

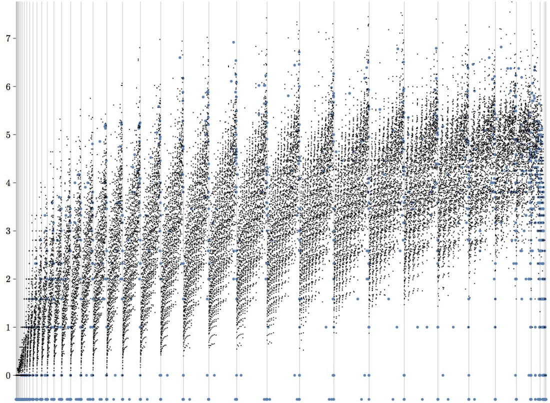
This is done used `=`, or in '`FullForm`', `Set`.

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 `Fib[5]` ⇝ `Fib[4] + Fib[3]`, and then `Fib[4]` ⇝ `Fib[3] + Fib[2]` and `Fib[3]` ⇝ `Fib[2] + Fib[1]`, so already we see `Fib[2]` will be evaluated twice.
- What we need is to force the evaluation of `Fib[4]` and `Fib[3]` and then store the result once and for all in `Fib[5]`. `Set` is ideal for this.
-
```
fib[0]  := 0
fib[1]  := 1
fib[n_] := fib[n] = fib[n-1] + 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.