Transcript: advantages of mutation

Transcript: advantages of mutation

- Efficiency
  - Space (esp. in call stack)
  - Time

Transcript: advantages of mutation

- Efficiency
  - Space (esp. in call stack)
  - Time
- Loops are easier to understand than recursion

Transcript: advantages of mutation

- Efficiency
  - Space (esp. in call stack)
  - Time
- Loops are easier to understand than recursion
- Memoisation is difficult to express declaratively

Transcript: disadvantages of mutation

Transcript: disadvantages of mutation

- Laziness causes problems

Transcript: disadvantages of mutation

- Laziness causes problems
- Von-Neumann bottleneck

Transcript: disadvantages of mutation

- Laziness causes problems
- Von-Neumann bottleneck
- Lack of referential integrity
  (Wegen Reihenfolgeabhängigkeit)

The book

Peter Van Roy, Seif Haridi.
Concepts, Techniques, and Models of Computer Programming.
MIT Press, March 2004.
Section 4.8, page 313.

Subjectivity

Different people may find different models easier to use, because of their differing knowledge and background. The issue is not the precise definition of *natural*, but the fact that such a definition exists for each person, even though it might be different for different people.

John Backus's Turing Award lecture

> The assignment statement is the von Neumann bottleneck of programming languages and keeps us thinking in word-at-a-time terms in much the same way the computer's bottleneck does.

Edsger W. Dijkstra's review

A naive implementation [of immutable matrices] that first copies those matrices and then kicks out half of it again seems absolutely unacceptable on any machine—von Neumann or not.

Edsger W. Dijkstra's review

A naive implementation [of immutable matrices] that first copies those matrices and then kicks out half of it again seems absolutely unacceptable on any machine—von Neumann or not.

`www.haskell.org/haskellwiki/GHC/Memory_Management`

Naturalness versus efficiency

A programme is *natural* if very little code is needed just for technical reasons unrelated to the problem at hand.

A natural Haskell programme

```
fib 0 = 1
fib 1 = 1
fib n = fib (n - 2) + fib (n - 1)
```

A fast Haskell programme

```
fib2 n =
 let loop 0 prev next = next
     loop m prev next = loop (m - 1) next (prev + next)
 in
     loop n 0 1
```

A matter of compiler intelligence?

It is unrealistic to expect the compiler to rewrite your programme. Even after several decades of research, no such compiler exists for general-purpose programming.

Functional means natural?

```
case Letrec(x, e, body) => {
  val mutableenv = scala.collection.mutable.Map() ++ env
  mutableenv += x -> eval(e, mutableenv)
  eval(body,mutableenv)
}
```

Functional means natural?

Circular lists in Haskell

`www.haskell.org/haskellwiki/Tying_the_Knot`

Simplicity: where?

The declarative model is one of the simplest of all. It has serious limitations for some applications. There are more expressive models that overcome these limitations, at the price of sometimes making reasoning more complicated.

Simplicity: where?

The more expressive models are not *better* than the others, since they do not always give simple programmes and reasoning in them is usually harder.