functional programming in practice

Freek Wiedijk

Radboud University Nijmegen

course ‘Principles of Programming Languages’

Free University Amsterdam

2006 05 11, 11:00

why I use functional languages

the story of LCF and ML

proof assistants = programs to help create correct mathematical proofs

research on proof assistants $\longrightarrow$ functional programming
$\longrightarrow$ emacs

functional programming = 'spin off' of proof assistant technology

Robin Milner $\longrightarrow$ Turing award in 1991

process algebra $\longrightarrow$ CCS $\longrightarrow$ π-calculus

proof assistants $\longrightarrow$ LCF proof assistant

'Logic of Computable Functions'

scripting language for LCF $\longrightarrow$ ML = 'meta language'

functional languages

- **lisp**

  prehistoric (1958), big, *untyped*

- **ML**

  typed, *strict*, supports imperative programming

  - **SML** = standard ML
  - **ocaml**

- **haskell**

  typed, *lazy*, purely functional

- **clean**

  ‘improved haskell’, made in Nijmegen

some functional programs

functional programs that I really use

- **advi**

  ‘active DVI’

  powerpoint-like presentation software for LaTeX

  presents a dvi-file with effects

- **unison**

  file synchronisation software

  keeps two file trees identical

  runs on Unix, Windows & Mac

the best proof assistants

• **HOL**	
– **HOL4**	$\longrightarrow$ SML
– **HOL Light**	$\longrightarrow$ ocaml
– **ProofPower**	$\longrightarrow$ SML
– **Isabelle**	$\longrightarrow$ SML
• **coq**	$\longrightarrow$ ocaml
• **PVS**	$\longrightarrow$ lisp & ocaml
• **ACL2**	$\longrightarrow$ lisp
• **mizar**	$\longrightarrow$ pascal

proof assistants that are programming languages

a proof assistant that is also a logic programming language

- **twelf** $\longrightarrow$ SML

proof assistants that want to be functional programming languages

- **agda** $\longrightarrow$ haskell
- **epigram** $\longrightarrow$ haskell

dependently typed functional programming

functional programming languages steadily become more abstract impractical?
lisp $\longrightarrow$ **ML** $\longrightarrow$ **haskell** $\longrightarrow$ coq / agda / epigram / ...

John Harrison's theorem provers

HOL Light

LCF $\longrightarrow$ HOL $\longrightarrow$ HOL Light

John Harrison

– verifies floating point hardware for Intel

– has verified the most theorems in the world

<`http://www.cs.ru.nl/~freek/100/`> or google for $\boxed{\text{freek 100}}$

HOL Light source

44 files = 25k lines = 1M source

in HOL there is no difference between programming and proving!

HOL proof = ML program that returns an object of datatype 'thm'

the theorem prover from John's book

Introduction to Logic and Automated Theorem Proving

currently 820 pages
to be published by Cambridge University Press

everything explained through code samples
all code samples together $\longrightarrow$ fully functional proof tool

<`http://www.cl.cam.ac.uk/users/jrh/atp/`>

or google for $\boxed{\text{theorem proving examples}}$

so why functional programming?

easy data

- algebraic datatypes + pattern matching

```
(* Type for recording history. *)
type history =
   Start of int
 | Mmul of (num * (int list)) * history
 | Add of history * history;;
```

- garbage collection

clean code

functional programming makes it . . .

- . . . much more difficult to get a program that even typechecks
- . . . much more difficult to get a program that has subtle bugs