Functional Programming at Linspire

Commercial Uses of Functional Programming
Portland, Oregon

Clifford Beshers
Linspire, Inc.
Linspire/Freespire

- Linux distribution for consumers
- No system manager
- Ultimately an appliance
  - Preserve flexibility
  - Improve reliability
  - Reduce complexity
History of FP at Linspire

- **David Fox**
  - Scheme -> O'Caml -> Haskell

- **Clifford Beshers**
  - Scheme -> O'Caml -> Haskell

- **Jeremy Shaw**
  - Clean -> O'Caml -> Haskell

- **Sean Meiners**
  - O'Caml -> Haskell
Why Haskell?

• Medical researchers announced today that MRI studies reveal that the structure of the human brain is significantly different in adolescence than adulthood.
Why Haskell?

• Medical researchers announced today that MRI studies reveal that the structure of the human brain is significantly different in adolescence than adulthood.

• *Well, duh!*
Why Shift from O'Caml to Haskell

- Type classes, IO Monad, function composition
- Complete libraries, Module system
- ghci/:help, ghc –make
- Readability, ML -> mass of nested lets
- Active community, growing momentum
- Better name
Haskell Performance

• Strings as thunks
  – Parsac ineffective on big files
  – Solved by ByteString (FPS)
  – Factor of 10 reduction in time and space

• Strictness forced by lots of IO

• Cross that bridge...
Programs in FP

- Original Warehouse (Debian → Click and Run)
- Hardware detector (System configuration) (O'Caml)
- ISO builder ([Package] → .. → CD Live/Install) (O'Caml)
- CGI – views of internal databases (O'Caml, Haskell)
- Package autobuilder and Debian library (Haskell)
  - In progress
- Miscellaneous scripts
Hardware Detection

• All drivers included in OS
• Detect & configure on every boot
  – Interchangeable drives
• Maintain legacy file formats
  – /etc/fstab, /etc/modules
• Goal: zero configuration on boot
Hardware Environment

- Unknown platform, unknown environment
  - Static type checking avoids run time errors in the field

- No user serviceable parts inside
  - Total functions, everything Maybe
  - Never give up, no exceptions
    - missing PCI bus about the only excuse for stopping
Rose Colored Glasses

- QA calls it the ``defector''
- Any glitch breaks entire system
- Many glitches
- Let's back out and take a look at the bigger picture
What I Learned from SCUBA

• Accident analysis
  – It's the third thing that goes wrong that kills you

• Doing It Right (DIR)
  – Train until muscle memory lets you fix any problem quickly
  – Always dive with a buddy who is DIR trained
  – Never take your eyes off the water

• Resentment from old hands
  – Until they take the course
Functional Mindset

- Not enough to just use functional languages
- Apply functional techniques everywhere possible
  - Analyze types
  - `Type check' as early as possible
  - Purely functional data structures
  - Test suites
  - Correctness first. (Pillage, then burn!!!)
    - Premature optimization is the root of all evil.
Building Install ISOs

• chroot $dir apt-get install $packages
• Pack into compressed loopback
• Total run time, 40 minutes plus 10 minute install
• Lessons from FP -- ``Type check'' early
  – Packages exist and dependencies satisfied
  – CD below 650MB
  – Throw exceptions at the slightest provocation
Wayback machine

- Functional data structures for backup
  - Copy local tree to backup directory on server, timestamp
  - Modify local tree
  - Copy most recent server backup tree to new timestamp
    - Hard link all files back to the original
  - Copy from local tree to new timestamp directory on server
- rsync does the work, breaks the hardlinks
- copy releases to precious directory, garbage collect
Source Code Management

- GNU Arch/tla
- Tree of patches
- Only operation is patch append
- Commits are permanent
- Undo by committing patch in reverse
V1/V2 Software Warehouses

- **V1**: O'Caml, but not functional implementation
  - Rapid development
  - Changing specs
  - Bad results

- **V2**: Perl, careful design and implementation, but...
  - Unnecessary centralization
  - Modularity not at tool level
  - Package management not functional
Functional Distribution Management

- Old: packages moved in groups from unstable \(\rightarrow\) stable
- New: patch model for distribution specification
  - A patch is a set of new source debs
  - autobuilder \(\text{:: [SrcDeb]} \rightarrow \text{IO [(SrcDeb,BinDeb)]}\)
  - fastbuilder \(\text{:: [(SrcDeb,BinDeb)]} \rightarrow \text{[SrcDeb]}
    \rightarrow \text{IO [(SrcDeb,BinDeb)]}\)
- As with source code, immutable trees of distributions
Legacy Applications

• Linux has old and new device paths
  – /dev/hda, /dev/hda1
  – /dev/ide/host0/bus1/target0/lun0/disc, .../part1

• grub takes only old format

• phantom typing, newtype

• values passed from O'Caml to bash to C++, back to bash, back to O'Caml and lost their type
Proc Filesystem Formats

- ASCII, Binary, CSV, Attr/Value, you name it.
- /proc/bus/usb/devices

T:  Bus=04 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1  Spd=12  MxCh=  2
B:  Alloc=  0/900 us ( 0%),  #Int=  0,  #Iso=  0
D:  Ver= 1.10  Cls=09(hub )  Sub=00  Prot=00  MxPS=  8  #Cfgs=  1
P:  Vendor=0000  ProdID=0000  Rev= 2.06
S:  Manufacturer=Linux 2.6.14 uhci_hcd
S:  Product=UHCI Host Controller
S:  SerialNumber=0000:00:1d.2
C:*  #Ifs= 1  Cfg#= 1  Atr=c0  MxPwr=  0mA
I:  If#=  0  Alt=  0  #EPs=  1  Cls=09(hub )  Sub=00  Prot=00  Driver=hub
E:  Ad=81(I)  Atr=03(Int.)  MxPS=  2  lvl=255ms
Shell Strengths and Weaknesses

- Dead simple file I/O
- Composition of functions
- Untyped
- Flat streams
- Unix tools not standardized
  - Options, formats, regex expressions
- Better dead than proficient at Perl
Replacing shells

- system $ unwords [cmd,flag,arg]  # bah
- Jeremy Shaw – Pipes.hs   -- bah!
  - recreation of shell process handling
- Replace simple commands with native functions
  - find, grep, sed, rm, mv, xargs,...
- Wrap big legacy programs in typesafe modules
- Strive for elegance of composition
Type Safe System Programming

- Each data format gets a type and a parser
- Each legacy program gets a module
- Each subcommand gets a function
- Each function gets data using DualForm
- Synchronizes data between memory and disk on demand
- Composition with standard IO Monad
- Minimal magic
Data Format

class FileFormat a where
  load :: String -> a
  save :: a -> String
Legacy Program: Debian Apt

type SourcesList = DualForm String
type DebianIndex = DualForm String

update :: Maybe SourcesList -> IO [DebianIndex]

install :: [PackageName] -> [DebianIndex] -> FilePath -> IO ()
DualForm

data (FileFormat a) => DualForm a =
    DualForm (IORef (Maybe FilePath)) (IORef (Maybe a))

fromValue :: (FileFormat a) => a -> IO (DualForm a)
fromFilePath :: (FileFormat a) => FilePath -> IO (DualForm a)

asFilePath :: (FileFormat a) => FilePath -> DualForm a -> IO FilePath
asValue :: (FileFormat a) => DualForm a -> IO a
DualForm Syncing

\[\text{asValue} :: (\text{FileFormat a}) \Rightarrow \text{DualForm a} \rightarrow \text{IO a}\]

\[\text{asValue} (\text{DualForm } \text{fpRef } \text{vRef}) = \]
\[\text{do } \text{mv } \leftarrow \text{readIOPRef } \text{vRef}\]
\[\text{case } \text{mv} \text{ of}\]
\[\text{(Just } \text{v}) \rightarrow \text{return } \text{v}\]
\[\text{Nothing } \rightarrow \]
\[\text{do } \text{mfp } \leftarrow \text{readIOPRef } \text{fpRef}\]
\[\text{case } \text{mfp} \text{ of}\]
\[\text{Nothing } \rightarrow \text{error "Panic: DualForm was empty."}\]
\[\text{(Just } \text{fp}) \rightarrow \]
\[\text{do } \text{v } \leftarrow \text{readFile } \text{fp} \rightarrow \text{return } (. \text{load})\]
\[\text{writeIOPRef } \text{vRef} \text{(Just } \text{v})\]
\[\text{return } \text{v}\]
import DualForm as DF
import DQL as DQL

mkSourcesList Debian.unstable >>=
    Debian.Apt.update >>=
    Apt.install packages

mkSourcesList Debian.unstable >>=
    Debian.Apt.update >>=
    DQL.select [DQL.Package, DQL.Version]
Replacing Shells, Part Two

- Memoization on disk (Make with checksums)
- Garbage collection of temporary files
- Standardized logging
- ShowXML
- Fault detection and traceback
  - Installer/hotplug bug
Linspire Focus

• Shifting away from core OS
• Back to software marketplace (Click and Run Warehouse)
• Focus on applications
• Functional languages, quite likely
• Functional mindset, absolutely
Summary

- Easy to gush about functional methods
- Solutions feel like minimal surfaces
- Haskell programming transcends interruptions
- Get more done, go home on time
- Functional mindset useful in any programming language