Demo: Mocha—An Objective-C Binding for Haskell

André Pang, University of NSW & CSIRO

andrep@cse.unsw.edu.au, Andre.Pang@csiro.au

August 30, 2003

What’s MochΛ?

Haskell to Objective-C/Cocoa bridge

- For Mac OS X
- Access Cocoa from Haskell

Implementation of my undergrad honours thesis

- Shows that the theory works
- More importantly, shows how to work the theory
- Haskell features: Existential data types, Template Haskell, scoped type variables, overlapping instances, multi-parameter type classes + functional dependencies
- Objective-C features: Run-time class hierarchy modification, invocation forwarding (surrogate objects), dynamic loading

Modelling OO in Haskell

Represent object-oriented class hierarchies in Haskell

- Achieved with type classes (Shields/SPJ/Meijer/Finne/…)

```
foo :: NSObject -> IO ()
bar :: SubNSObject o => o -> IO ()
```

- 100% type inference and type checking
- Convenient upcasting and downcasting

```
let array = downcast object :: NSArray
```

- Automated class hierarchy generation via Template Haskell: no need for a separate interface generator tool

Using Cocoa from Haskell

Four-line URL downloader in Haskell:

- Yes, this code actually works …

```
-- get command-line arguments
args <- System.getArgs; let (arg1:_) = args
-- make a new URL object
url <- _NSURL_ # urlWithString arg1
-- fetch the URL's contents as a String
urlData <- _NSString_ # stringWithContentsOfURL url
-- print out contents of URL
putStr urlData
```

MochΛ Demo

Cocoa kicks arse!

Direct Messaging

OO-style overloading (multimethods) in Haskell:

- Via multi-parameter type classes & functional dependencies
- Use a ‘phantom parameter’ for method name

```
class DirectMsg rcvr method arg reply
   | rcvr method arg -> reply
data Name_insert
instance SubNSObject o => DirectMsg
   ArrayObj Name_insert (Int, o) ()
instance SubNSObject o => DirectMsg
   HashtableObj  Name_insert o ()
insert :: DirectMsg rcvr Name_insert args reply =>
   args -> rcvr -> IO reply
```

I love MPTCs

Upcasting and Downcasting:

- Again via multi-parameter type classes:

```
class Cast sub super where
 upcast :: sub -> super; downcast :: super -> sub
instance Cast Coffee Drink where
 upcast Coffee = Drink; downcast Drink = Coffee
```

MochΛ Internals

From startup to finish:

1. Normal C `main()` or Haskell-written `main` invoked
2. Objective-C RTS loads HSProxy class
3. HSProxy searches executable for `__mocha_*` symbol names (`foreign export`ed Haskell functions, generated by Template Haskell) and executes them
4. The `__mocha_*` functions tell HSProxy to pose as the intended class of the Haskell module (e.g. MyObject)
5. HSProxy serves as *surrogate* object—forwards messages sent to it to Haskell functions
6. Haskell functions call back into Objective-C to do stuff
7. HSProxy forwards results from Haskell functions back to Objective-C RTS to the caller

Haskell on Mac OS X

Haskell as a supported language on Mac OS X

- Include Haskell compilers with Apple's (free) Developer Tools CD
- Already ships with Perl, Python, Tcl, Ruby …
- Apple keen to support Cocoa integration
  - PyObjC author now works for Apple
  - Perl/ObjC binding ships with Mac OS X
- Apple already uses niche technology (Objective-C, Mac platform in general) → open mindset is already there

Getting there

- GHC already in OpenDarwin
- GHC needs to have a smaller footprint
- Buy lots of beer for Jordan Hubbard

Thank You!

Where to get Mocha and more info

- http://www.algorithm.com.au/mocha